MASTER EN SCIENCES INFORMATIQUES

Programmation linéaire conviviale sur micro-ordinateur

Gerard, Annick

Award date:
1987

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix

Institut d'Informatique

PROGRAMMATION LINEAIRE CONVIVIALE SUR MICRO-ORDINATEUR

Annick GERARD

Promoteur : Monsieur **J-P. LECLERCQ**

Mémoire présenté en vue de l'obtention du grade de Licenciée et Maître en Informatique par **Annick GERARD**

année académique 86-87

Qu'il me soit permis ici de remercier Monsieur J-P. Leclercq pour l'intérêt avec lequel il a dirigé ce travail tout au long de l'année.

Je remercie également mon mari dont la gentillesse et la compréhension m'ont aidée à mener à bien l'ensemble de mes études.

TABLE DES MATIERES

CHAPITRE 0 : INTRODUCTION

0. INTRODUCTION

Le but de ce mémoire est la réalisation d'un programme permettant la résolution de problèmes d'optimisation linéaire ainsi que diverses manipulations sur les résultats obtenus, dans un esprit de convivialité susceptible de donner naissance à un produit d'utilisation aisée et agréable, en particulier au niveau de l'introduction des données. Nous présentons tout d'abord quelques considérations d'ordre général dans le but de cerner globalement l'apport de ce travail.

Le problème général de la programmation linéaire [DAN 66], [SIM 72] s'écrit :

$$\text{optimiser } z = \sum_{j=1}^{n} c_j \, x_j$$

sous des contraintes :

$$\sum_{j=1}^{n} a_{ij} \, x_j >= b_i \qquad i = 1, \ldots, p.$$

$$\sum_{j=1}^{n} a_{ij} \, x_j <= b_i \qquad i = p+1, \ldots, q.$$

$$\sum_{j=1}^{n} a_{ij} \, x_j = b_i \qquad i = q+1, \ldots, m.$$

$$x_j >= 0 \qquad j = 1, \ldots, r.$$

$$x_j \text{ quelconque} \qquad j = r+1, \ldots, n.$$

Les a_{ij}, b_i, c_j ($i = 1, \ldots, m$; $j = 1, \ldots, n$) sont des constantes données.

Ces expressions synthétiques peuvent être le reflet de formulations générales complexes.

Considérons à titre d'exemple le problème simplifié suivant [BRO] : un ensemble d'équipements donné, composé de machines et d'outils, est destiné à la production de différentes pièces. La production de chacune de ces pièces nécessite essentiellement une opération particulière, requérant l'utilisation d'un outil précis et de l'une des machines . En outre, production et ventes doivent être planifiés sur différentes périodes de temps.

Sont à déterminer les grandeurs suivantes :

- pour chaque période de temps, quelle quantité de chaque pièce produire.
- pour chaque période de temps, quelle quantité de chaque pièce vendre.
- pour chaque pièce, quelle quantité restante reporter dans le stock d'une période sur la suivante.
- pour chaque période de temps, quel est le nombre de jours de production de chaque outil en conjonction avec chaque machine.

Le but est de maximiser le profit total réalisé sur toutes les pièces pour la période de planification. Cette grandeur se calcule en tenant compte des éléments suivants :

- contribution au profit net de chaque pièce vendue
- coûts variables associés à la production
- coûts de stockage de chaque pièce

NB : on néglige les coûts fixes associés à la production

Les contraintes suivantes doivent être satisfaites :

- pour chaque période de temps, disponibilité limitée des outils.
- pour chaque période de temps, disponibilité limitée des machines.
- pour chaque période de temps et pour chaque pièce, les ventes ne peuvent excéder la demande prévue.
- pour chaque période de temps et pour chaque pièce, les ventes doivent excéder une certaine fraction de la demande prévue.
- pour chaque pièce, la quantité restante en fin de période de planification doit prendre une valeur précise.
- toute demande qui n'a pu être satisfaite est perdue.

Ce problème peut être mis sous forme de programme linéaire. Définissons tout d'abord le concept de *jour standard*. Il s'agit, pour une pièce donnée, de la quantité qui serait produite en un jour si un outil requis fonctionnait normalement sur une machine. Les conventions de notation seront les suivantes :

- indices :

  . i : indice les pièces

  . j : indice les outils

  . k : indice les machines

  . t : indice les périodes de temps, t = 1,...,T

- données :

  . a_{jt} : nombre de jours de disponibilité de l'outil j pendant la période t

  . b_{kt} : nombre de jours de disponibilité de la machine k pendant la période t

  . c_{jkt} : coût variable journalier associé à la production sur un outil j, sur une machine k et pendant la période t

  . d_{it} : demande prévue pour la pièce i pendant la période t, en jours standards

  . h_{it} : coût de stockage d'un jour standard de la pièce i pendant toute la durée de la période t

  . I_{i0} : stock initial de la pièce i, en jours standards (>= 0)

  . I_{iT} : stock final de la pièce i , en jours standards (>= 0)

  . p_{it} : contribution au profit de la vente d'une valeur d'un jour standard de la pièce i pendant la période t

  . s_{it} : fraction minimale de d_{it} qui doit être satisfaite ($0 <= s_{it} <= 1$)

- variables de décision :

. I_{it} : quantité prévue en stock de la pièce i en fin de période t, en jours standards (1 <= t < T)

. E_{it} : ventes prévues de la pièce i pendant la période t, en jours standards

. V_{jt} : jours de production prévus pour un outil j pendant la période t

. W_{jkt} : jours de production prévus pour un outil j sur une machine k pendant la période t

. U_{it} : production prévue de la pièce i pendant la période t, en jours standards

En utilisant ces notations, nous pouvons poser le problème sous la forme :

$$\text{Max} \sum_{it} p_{it}E_{it} - \sum_{jkt} c_{jkt}W_{jkt} - \sum_{it} h_{it}(I_{it-1} + I_{it})/2$$

$$\forall\, j,t \quad \sum_{k} W_{jkt} = V_{jt}$$

$$\forall\, j,t \quad \sum_{i} U_{it} = V_{jt}$$

$$\forall\, k,t \quad \sum_{j} W_{jkt} <= b_{kt}$$

$$\forall\, i,t \quad I_{it} = I_{it-1} + U_{it} - E_{it}$$

$$\forall\, i,t \quad s_{it}\, d_{it} <= E_{it} <= d_{it}$$

$$\forall\, j,t \quad 0 <= V_{jt} <= a_{jt}$$

$$\forall\, i \quad I_{it} >= 0 \qquad 1 <= t < T$$

$$\forall\, j,k,t \quad W_{jkt} >= 0$$

NB : on considère qu'aucune limite n'existe sur la quantité i de pièces productibles pendant une période.

Cet exemple permet d'imaginer combien peut être lourde et fastidieuse l'introduction en mémoire du problème, et particulièrement des données, lorsque l'on désire effectuer le traitement au moyen de l'un des logiciels de résolution existants. Ceux-ci ont en effet comme caractéristique commune une grande rigidité au niveau de la reconnaissance des données, ce qui exige de l'utilisateur une définition du problème spécialement destinée au logiciel et éloignée de la perception naturelle qui est la sienne, en raison de la multiplicité et de la forme des déclarations et des valeurs à introduire. Le logiciel LAMPS [AMSL], disponible aux Facultés en est un exemple bien qu'il propose un générateur de problèmes (MAGIC) essentiellement destiné à faciliter le traitement de très gros problèmes moyennant l'apprentissage d'un langage d'écriture [FOR].

Nous avons choisi de répondre à cet inconvénient par l'implémentation d'un logiciel capable de reconnaître les données d'un problème de programmation linéaire simple formulé dans une syntaxe proche de l'écriture mathématique naturelle de ce problème, telle qu'elle se ferait sur papier, en permettant en particulier l'utilisation du symbole de sommation et du quantificateur universel. L'utilisateur se contente donc d'introduire les données à l'aide d'un éditeur dont le choix lui revient. Cet ensemble de données est ensuite traité de manière à en corriger la syntaxe et à en construire une représentation adéquate dans la mémoire de l'ordinateur, représentation qui est finalement

transformée afin de permettre l'application immédiate de la méthode du simplexe.

En outre, un effort particulier a été fourni au niveau de l'analyse de sensibilité et du paramétrage dans le but d'offrir à l'utilisateur un choix simple et rapide des valeurs à traiter ainsi qu'une visualisation simple et claire des résultats obtenus. Il nous faut cependant signaler que le module de paramétrage annoncé ici comme dans la suite du texte, n'a pu faire l'objet de tests suffisants pour garantir une fiabilité satisfaisante. Par conséquent, tout accès à ce module a été supprimé dans la version actuellement disponible du logiciel.

Enfin, l'ensemble du travail, implémenté en langage Turbo Pascal, sur IBM PC, a été conçu dans un esprit de modularité susceptible de favoriser une extension future du projet : optimisation des performances, visualisation graphique de certains résultats, programmation linéaire en variables entières, partiellement ou totalement, programmation quadratique, ...

La présentation de ce texte est organisée comme suit.

Les deux premiers chapitres décrivent respectivement la syntaxe et sa représentation en mémoire. Si l'utilisateur peut considérer le premier de ces chapitres comme une sorte de guide ou de mode d'emploi lui permettant un usage satisfaisant du logiciel, il ne manquera pas de consulter le second s'il s'intéresse de plus près à la représentation des données en mémoire.

Le troisième chapitre propose un rappel théorique sur la programmation linéaire et explicite les points de théorie implémentés.

Le quatrième chapitre indique quelles sont les structures nécessaires à l'implémentation du programme linéaire proprement dit et envisage l'obtention de ces dernières à partir des structures générées par la reconnaissance de la syntaxe.

Le cinquième chapitre reprend les caractéristiques plus concrètes de l'implémentation, en particulier les limites imposées au niveau de la taille des données et leurs conséquences pratiques au niveau de l'utilisation optimale du logiciel.

A la suite du chapitre consacré à la conclusion, en annexe, seront exposés d'une part une description des procédures et fonctions, au sein du listing du programme, et d'autre part un compte-rendu des tests effectués.

L'exemple développé plus haut nous servira de fil conducteur tout au long du texte et nous permettra d'illustrer avantageusement les différents points abordés.

CHAPITRE 1 : DEFINITION DE LA SYNTAXE

1. DEFINITION DE LA SYNTAXE

1.1 GENERALITES

Comme indiqué précédemment, l'accent a été mis ici sur la convivialité. En effet, moyennant quelques restrictions, l'utilisateur peut introduire un problème de programmation linéaire simple au clavier sous une forme très proche de l'écriture mathématique de ce problème, les précautions à prendre étant introduites pour le respect d'une logique inhérente à l'utilisation d'un ordinateur. Attirons immédiatement l'attention sur le fait qu'en plus des symboles de quantification universelle et de sommation, certaines notations mathématiques anodines demandent à être adaptées afin d'être aisément reconnues par le logiciel tout en gardant leur sémantique. L'exemple développé dans le chapitre introductif illustre ce point : quantification universelle sur toutes les valeurs d'indices à l'exception d'une seule, contraintes supplémentaires sur les variables pour une valeur particulière d'indice, considérations de décalage sur l'ensemble des valeurs d'indices ($I_{i,t}/2 + I_{i,t-1}/2$). Des solutions particulières ont dû être développées et sont présentées plus bas

En outre, le logiciel offre à l'utilisateur la possibilité de visualiser les résultats obtenus à l'écran, d'en garder trace écrite au moyen d'une impression ou de les mémoriser sur disque, toute combinaison de choix étant possible.

Par contre, aucune possibilité n'étant prévue pour déclarer le signe des variables intervenant dans le problème, celles-ci sont supposées positives, ce qui a comme conséquence d'obliger éventuellement l'utilisateur à aménager l'expression du problème avant tout usage de ce logiciel.

La première apparition d'un nom de variable et de ses éventuels indices (noms de familles) au sein de la fonction objectif ou d'une contrainte tient lieu de déclaration pour cette variable.

En toute généralité, l'utilisateur doit introduire les données en quatre paragraphes distincts :

1. déclaration des familles d'indices
2. déclaration des constantes
3. écriture de la fonction objectif et des contraintes
4. commentaires relatifs à l'application

Seul le troisième de ces paragraphes est obligatoire mais si plus d'un paragraphe apparaît, l'ordre indiqué ci-dessus doit être respecté.

Chaque paragraphe, s'il apparaît, doit commencer par un identificateur suivi du séparateur ":", respectivement

1. Fam
2. Cst
3. Equ
4. Com

Le premier paragraphe contient de *1* à *15* déclarations de familles d'indices différentes séparées par ";".

Le second rassemble de *1* à *15* déclarations de constantes différentes séparées par ";".

Le troisième regroupe la *fonction objectif* et de *1* à *30* contraintes sous forme d'équations ou d'inéquations, toujours séparées par ";".

Cependant, pour des raisons techniques qui seront évoquées en détail dans le chapitre traitant des caractéristiques relatives à l'implémentation, l'utilisateur est invité à veiller à ce que le problème introduit ne conduise pas à un tableau du simplexe dont la taille dépasse *50 lignes* sur *90 colonnes*, ces dimensions incluant la démultiplication éventuelle et automatique des contraintes après explicitation des quantificateurs universels ainsi que les variables d'écart et/ou artificielles automatiquement ajoutées en cas de nécessité. En pratique, le fonctionnement correct du logiciel est garanti tant qu'est satisfaite la formule suivante :

*90 = nombre de variables + 2 * nombre de lignes*

Ajoutons enfin qu'en aucun cas le nombre de *noms* de variables ne dépassera *20*.

Une déclaration, la fonction objectif et les contraintes peuvent s'étendre sur plusieurs lignes, la longueur de celles-ci étant limitée par l'éditeur choisi.

Les autres séparateurs susceptibles d'intervenir dans une déclaration sont :

"(", ")", "[", "]", ",", "_", "+", "-", "*", "=".

Une suite de caractères terminée par un séparateur ne peut avoir une longueur excédant 80 caractères, quelle que soit sa fonction.

La liste des mots réservés est la suivante :

Fam, Cst, Equ, Com, Fin, Som.

L'usage des quatre premiers de ces mots a été vu plus haut. "Fin" suivi du séparateur ":" signale la fin de l'introduction du problème. Toute information introduite à la suite de "Fin :" sera ignoréée. "Som" est le symbole de sommation.

Sont également réservés les symboles ">=", "<=" et "#", ce dernier étant choisi pour représenter le quantificateur universel.

Les noms de familles d'indices, de constantes, de contraintes et de variables doivent être alphanumériques, uniques (identifiants), ne contenir aucun séparateur et être différents des mots et symboles réservés.

Le logiciel implémenté ici se limite à 10 caractères identifiants. Majuscules et minuscules ne sont pas différenciées.

1.2 CONVENTIONS PRELIMINAIRES

Dans le but de faciliter la compréhension des règles de syntaxe détaillées qui vont être données ci-dessous, nous établissons les conventions suivantes :

1. - un ensemble d'éléments entre accolades suivies d'un indice doit être répété selon les valeurs de cet indice.

- un "/" entre deux éléments indique un choix possible entre ces éléments.

2. *<familles>* désigne la chaîne

{<nom de famille>}$_M$ (,<nom de famille>}$_N$

0 <= M <= 1

0 <= N <= 2

Exemples : après avoir déclaré I, J et K ,on peut écrire :

(aucun nom de famille)

I

I, J

I, J, K

3. Si plusieurs indices sont présents, l'introduction des valeurs de constantes se fait en considérant que l'indice apparaissant le plus à droite varie le plus rapidement. En particulier, toute matrice est introduite ligne par ligne.

4. Au sein de toute déclaration de constante, une attention particulière doit être accordée au nombre de valeurs réelles introduites après le signe d'égalité. En effet, l'utilisation du quantificateur universel, noté "#", permet de ne pas indiquer explicitement toutes les valeurs mais en exige le nombre nécessaire et suffisant pour garantir la cohérence de la déclaration.

Exemples : après avoir déclaré

```
I = [1_3];
J = [1_2];
```

on peut écrire : `#I, C1(I,J) = 1, 2;`

`#I, #J, C2(I,J) = 3;`

5. *<famille sauf valeur>* désigne la chaîne

<nom de famille> <> *<valeur>*

où *<valeur>* doit être :

- soit un *entier* appartenant à l'intervalle définissant la famille *<nom de famille>* si celle-ci est déclarée par *intervalle*
- soit une *"valeur"* appartenant à l'énumération définissant la famille *<nom de famille>* si celle-ci est déclarée par *énumération* (voir plus bas ces deux possibilités).

Exemples : après avoir déclaré

```
I = [1_3];
ville = (namur, bruxelles);
```

on peut écrire : `I <> 2`

`ville <> bruxelles`

5b. *<familles exceptions>* désigne la chaîne

{<famille sauf valeur>/<nom de famille>}$_M$
{,<famille sauf valeur>/,<nom de famille>}$_N$

0 <= M <= 1

0 <= N <= 2

Exemples : après avoir déclaré

I = [1_3];

J = [1_4];

ville = (namur, bruxelles);

on peut écrire :

I <> 1

I, J <> 2

I, J <> 3, ville <> namur

I, J, ville

I <> 2, J, ville

6. *<fam pour val glob>* désigne la chaîne

<nom de famille> >< <valeur globale>

où *<valeur globale>* doit être :

- soit *<valeur>* au sens de la convention précédente

- soit un *<nom de famille>*, *"><"* exprimant une *égalité généralisée* à ce dernier cas, signifiant que les valeurs successives prises par les familles situées de part et d'autre du signe *"><"* progressent *en parallèle.* Seuls des familles définies *par intervalle* (voir ci-dessous le paragraphe *FAMILLES D'INDICES*) peuvent apparaître de part et d'autre de ce signe.

Exemples : après avoir déclaré

I = [1_3];

ville = (namur, bruxelles);

temps = [1_12];

temps décalé = [0_11];

on peut écrire :

I >< 3

ville >< bruxelles

temps >< temps décalé

6b. *<familles égalités>* désigne la chaîne

{<fam pour val glob>/<nom de famille>}$_M$
{,<fam pour val glob>/,<nom de famille>}$_N$

0 <= M <= 1

0 <= N <= 2

Exemples : après avoir déclaré

I = [1_3];

J = [0_2];

ville = (namur, bruxelles);

on peut écrire :

I, J >< 2

I >< 1, ville >< namur

I >< J, ville

I, J >< 0, ville

Remarque : une illustration explicite de l'utilité des mécanismes introduits dans les conventions 5 à 6b sera donnée plus bas dans le paragraphe *EXEMPLE*, où nous retrouverons les problèmes particuliers introduits en début de chapitre reflétés par l'exemple introductif.

7. *<élément de somme>* désigne la chaîne

*{<réel signé>}$_M$ * Som(<familles exceptions>)*
*{<réel non signé>}$_N$ **
*<nom de constante>(<familles égalités>) **
<nom de variable>(<familles égalités>)

0 <= M <= 1

0 <= N <= 1

Au sein même de <élément de somme>, des règles de cohérence sont à respecter :

- "Som" est facultatif.

- si "Som" apparaît, il doit être suivi d'une parenthèse ouvrante et d'au moins un nom de famille ainsi que d'au moins le champ <nom de variable>.

- dans la déclaration de la fonction objectif, tout nom de famille apparaissant éventuellement entre les parenthèses qui suivent le <nom de constante> et/ou le <nom de variable>, à gauche du symbole "><", doit apparaître dans les parenthèses dépendant de "Som" si celui-ci est présent, sauf si le symbole "><" est suivi d'une valeur particulière dans chacun des champs apparaissant dans la déclaration.

- dans la déclaration d'une contrainte, tout nom de famille apparaissant éventuellement entre les parenthèses qui suivent le <nom de constante> et/ou le <nom de variable>, à gauche du symbole "><", doit apparaître soit dans les parenthèses dépendant de "Som", si celui-ci est présent, sauf si le symbole "><" est suivi d'une valeur particulière dans chacun des champs apparaissant dans la déclaration, soit dans l'ensemble du ou des nom(s) de famille apparaissant après d'éventuels "#".

- les opérations arithmétiques ne sont pas permises entre réels ni entre constantes au sein d'un même <élément de somme>.

- on remarquera que les sommes (Som) imbriquées doivent être transformées : Som(I) Som(J) doit s'écrire Som(I, J).

Exemples : après avoir déclaré les familles et constantes nécessaires, on peut écrire :

3 * som(i,j<>1) 2 * cst1(i,j) * var1(i,j)

-5 * cst2(i >< 1,j,k) * var2(i >< 1,j,k)

4 * x

cst3

som(i,j) cst1(i >< 1,j) * var1(i,j)

som(j) cst1(i >< 1,j) * var1(i >< 1,j)

som(i,j) cst4(i >< 1,i) * var1(i >< 1,j)

8. *<élément sans variable>* désigne la chaîne

*{<réel signé>}$_M$ * Som(<familles exceptions>)*
*{<réel non signé>}$_M$ **
<nom de constante>(<familles égalités>)

0 <= M <= 1

0 <= N <= 1

Des conventions analogues à celles exposées ci-dessus sont à respecter.

Exemples : après avoir déclaré les familles et constantes nécessaires, on peut écrire :

- 3 * som(i<>1,j) cst1(i,j)

4 * cst2(i)

9. *<sep>* désigne l'un des symboles ou groupes de symboles

"=", ">=", "<="

1.3 FAMILLES D'INDICES

Sont permises

1. les familles définies par intervalle :

<nom de famille> = [<entier>_<entier>];

Dans le cas d'une telle déclaration, le logiciel refuse une borne inférieure supérieure ou égale à la borne supérieure.

2. les familles définies par énumération :

<nom de famille> = (<nom$_1$>, ..., <nom$_N$>);

N >= 2

Les noms intervenant dans l'énumération peuvent être des entiers, des réels ou des chaînes de caractères mais n'apparaître qu'une seule fois au cours de la même énumération.

Exemples de déclarations correctes :

```
Fam : I = [1_20];
      J = [-5_5];
      origine = (Liège, Namur, Bruxelles);
      destination = (Namur, Anvers);
      impair = (un, trois, cinq, sept);
```

Exemples de déclarations erronées :

```
Fam : K = [5_1];
      ville = (Liège, Namur, Liège);
      impair = (1, 3, 5, 7);
```

1.4 CONSTANTES

Les constantes peuvent avoir 0, 1, 2 ou 3 indice(s) et la déclaration des constantes avec indice(s) peut être facilitée par l'utilisation du quantificateur universel qui dans ce cadre sera noté "#".

Toute déclaration de constante est de la forme suivante :

{#<nom de famille>,}$_A$ <nom de constante>(<familles>) = <réel>{,<réel>}$_B$;

0 <= A <= 3

B étant tel qu'il obéit à la quatrième convention préliminaire.

Exemples de déclarations :

Supposons les déclarations de familles suivantes :

```
fam : I = [1_2];
      J = [1_3];
      K = [1_4];
```

Sont correctes les déclarations de constantes :

```
cst : Scalaire = 25;
      Vecteur(I) = 1.5, 2;
      Matrice(I, J) = 1, 2, 3.5, 4, 5, 6;
      #I, Tab1(I) = 1;
      #I, Tab2(I, J) = 1, 2, 3;
      #I, #K, Tab3(I, J, K) = 1, 2.5, 3;
```

Sont erronées les déclarations :

```
cst : Scalaire = 1, 2;
      Vecteur(I) = 1.5;
      #I, Tab1(I) = 1, 2;
      #J, Tab1(I) = 1;
```

L'utilisateur dispose en outre de la possibilité de réinitialiser une ou plusieurs des valeurs de la constante *déclarée sous le nom le plus récent*, un nombre infini de fois, *seule la ou les dernière(s) valeur(s) introduite(s) étant prise(s) en compte.* Les règles de respect de cohérence vues au niveau des conventions préliminaires restent bien sûr d'application.

Exemples :

Supposons les mêmes déclarations de familles que plus haut.

Sont correctes les déclarations :

```
cst : Scalaire = 25;
      Scalaire = 30;
      #I, Vecteur(I) = 1;
      Vecteur(2) = 2;
      #I, #J, Matrice(I, J) = 1;
      #I, Matrice(I, J) = 1, 2, 3;
      Matrice(2, 3) = 7;
```

Est erroné l'ensemble formé des déclarations :

```
cst : Scalaire = 25;
      Vecteur(I) = 1, 2;
      Scalaire = 30;
```

1.5 FONCTION OBJECTIF ET CONTRAINTES

La première déclaration de ce paragraphe est en fait la fonction objectif. Elle doit être de la forme :

Min/Max [{<élément de somme>/
<élément sans variable>}$_A$];

1 <= A <= 7

Il est bien évident que si le premier élément est positif, son signe est facultatif.

La forme de la déclaration d'une contrainte est :

<nom de contrainte>, {#<famille sauf valeur>,}A
{<élément de somme>}B <sep>
{<élément sans variable>}C ;

0 <= A <= 3

1 <= B <= 7

1 <= C <= 7 - B

Exemples :

Les déclarations des noms de familles et de constantes étant celles du premier exemple du paragraphe précédent, seront considérées comme correctes les déclarations de fonction objectif et de contraintes suivantes :

```
Min [3 * X - 2.5 * Y + 7 * Z];

Un, X - 3 * Y >= 10;

Deux, - 4 * Y + 2.5 * Z >= 8.5;

Max [-3 * Som(I) Vecteur(I) * Var1(I) +
      2 * Scalaire +
      Som(I, J) 4 * Matrice(I, J) * X(I, J)];

Contr1, #J, Som(I) Vecteur(I) * X(I, J) <=
             Som(I) Tab2(I, J);

Contr2, #I, Vecteur(I) * Var1(I) + Var2 +
            Som(J) Matrice(I, J) * X(I, J) <=
            Tab1(I) + Som(J) Tab2(I, J);
```

1.6 COMMENTAIRES

Ce paragraphe est destiné à permettre à l'utilisateur la constitution d'un ensemble d'informations relatives à l'application traitée :

signification des variables, constantes et contraintes, contexte, etc. Il est constitué d'un texte libre sur lequel ne porte aucune vérification.

1.7 EXEMPLE

Afin de mettre en évidence les restrictions et avantages inhérents à l'utilisation du logiciel, nous donnons ici une version de l'exemple développé dans l'introduction dont la syntaxe répond à l'ensemble des exigences présentées ci-dessus. Les valeurs explicites indiquées ne le sont qu'à titre d'exemple et ont été choisies essentiellement pour illustrer au maximum les caractéristiques du logiciel.

```
Fam : i = [1_10];
      j = [1_3];
      k = [1_5];
      t = [1_12];
      tp = [0_11];

Cst : #t, a(j, t) = 3, 4, 5;
      #t, #k, b(k, t) = 4;
      #t, b(3, t) = 3;
      #j, #t, c(j, k, t) = 3.2, 4.5, 6, 2.1, 4;
      #t, d(i, t) = 3, 2, 3, 2, 4, 2, 3, 3, 4, 4;
      d(4, 4) = 4;
      #i, h(i, t) = 1, 2, 3, 4, 5, 6,
                    7, 8, 9, 10, 11, 12;
      #i, h(i, 4) = 0;
      #i, init(i) = 0;
      init(3) = 1;
```

```
init(5) = 3;

init(7) = 2;

#i, fin(i) = 1;

#t, p(i, t) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;

#t, #i, s(i, t) = 0.85;

Equ : Max[ Som(i, t) p(i, t) * E(i, t) -
          Som(j, k, t) c(j, k, t) * W(j, k, t) -
          0.5 * Som(i) h(i, t><1) * varinit(i) -
          0.5 * Som(i, t<>1) h(i, t) * II(i, t><tp) -
          Som(i, t) 0.5 * h(i, t) * II(i, t) ];

prod1, #j, #t, Som(k) W(j, k, t) - V(j,t) = 0;

prod2, #j, #t, Som(i) U(i, t) - V(j, t) = 0;

dispo1, #k, #t, Som(j) W(j, k, t) <= b(k, t);

dispo2, #j, #t, V(j, t) <= a(j, t);

equi1, #i, II(i,t><1) - U(i,t><1) + E(i,t><1) = init(i);

equi2, #i, #t<>1, II(i,t)-II(i,t><tp)-U(i,t)+E(i,t) = 0;

final, #i, II(i, t><12) = fin(i);

initial, #i, varinit(i) = init(i);

fraction, #i, #t, vars(i, t) = s(i, t);

vente1, #i, #t, E(i, t) - d(i, t) * vars(i, t) >= 0;

vente2, #i, #t, E(i, t) <= d(i, t);
```

Com : Détaillons quelque peu certaines déclarations de constantes afin de fixer les idées :

- la forme de la déclaration de a(j, t) a pour effet la mémorisation de la matrice suivante :

```
3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5
```

- les deux lignes constituant la déclaration de b(k, t) induisent la matrice :

```
4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4
3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4
4 4 4 4 4 4 4 4 4 4 4 4
```

- enfin, la déclaration de h(i, t) constitue la matrice :

```
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
1 2 3 0 5 6 7 8 9 10 11 12
```

Au niveau des déclarations de contraintes et de la fonction objectif, on remarquera les restrictions suivantes :

- introduction du réel sous forme décimale.

- "démultiplication" d'une même expression mathématique au niveau de la fonction objectif. En effet, la constante du problème I_{10} ne peut apparaître ici en tant que cas particulier d'une variable I_{1t}, notée I(i,t), t n'étant défini que de 1 à 12. Il y a donc obligation d'introduire une famille de "décalage" (tp) et d'identifier autrement I_{10}. Les équations "equi1", "equi2" sont le résultat d'un raisonnement analogue et l'équation "final" impose à I_{112} une valeur constante.

- nécessité de changer le nom de la variable "I", en l'occurence en "II", en raison de l'exigence de noms identifiants et de l'existence préalable d'une famille d'indices de même nom.

- introduction de la variable "varinit" en lieu et place de la constante "init" dans la fonction objectif et de la variable "vars" pour la constante "s" dans l'équation "vente1", introduction due aux restrictions de syntaxe au sein de <élement de somme> (on ne peut multiplier une constante que par un réel ou une variable).

- en corollaire, introduction des équations "initial" et "fraction" pour préserver la sémantique du problème.

- obtention des deux inéquations "vente1" et "vente2" due à l'obligation de n'avoir qu'un seul signe d'(in)égalité par déclaration d'(in)équation.

- dans une contrainte, report de tous les éléments de somme contenant au moins une variable à gauche du séparateur et de tous ceux n'en contenant pas à droite.

Fin :

1.8 QUELQUES INDICATIONS UTILES

Il ne s'agit pas ici d'un mode d'emploi proprement dit, le logiciel guidant l'utilisateur aux travers des différentes phases du travail. Nous nous contenterons donc de résumer ici la chronologie des actions à réaliser.

Après avoir créé un fichier de données au moyen d'un éditeur dont le choix est libre, l'utilisateur demande l'exécution du programme par la commande <prolico>

(PROgramme LInéaire COnvivial). Apparaît alors un écran par l'intermédiaire duquel lui sont demandés :

- le nom du fichier de données à traiter
- les périphériques de sortie souhaités :
  . écran (o/n)
  . imprimante (o/n)
  . disque (nom du fichier de sortie/<vide>)

Notes : - les valeurs par défaut pour les périphériques de sortie sont respectivement :

. o

. n

. <vide>

- toute combinaison de choix est possible mais un périphérique au moins est exigé.

- lorsque l'utilisateur est passé sur tous les champs, un accord définitif lui est demandé. En cas de réponse négative de sa part, il doit parcourir à nouveau tous les champs. En particulier, l'annulation d'un fichier de sortie se fait au moyen des symboles "//".

Des messages d'erreur sont générés si le fichier de données à traiter n'existe pas ou si aucun périphérique de sortie n'a été choisi. L'écran est ensuite réaffiché et l'utilisateur a la possibilité d'introduire d'autres réponses aux questions.

Lorsque le logiciel peut traiter complètement le fichier, il rend les résultats à l'utilisateur sur le(s) périphérique(s) demandé(s). Un nouvel écran est ensuite affiché pour permettre à l'utilisateur de poursuivre par

une analyse de sensibilité et/ou un paramétrage, sur les coûts et/ou sur les seconds membres des contraintes. En cas de réponse positive à l'une au moins de ces questions, l'utilisateur est invité à choisir les coûts et/ou les seconds membres par l'intermédiaire de la ou des variable(s) qui y correspondent. Toutes les demandes sont enregistrées à ce moment et seront traitées successivement, le logiciel se limitant à l'analyse de sensibilité et au paramétrage sur un seul élément à la fois.

Notes : - si l'utilisateur a souhaité un paramétrage, il lui sera également demandé d'introduire les bornes de l'intervalle pour chaque variable pour laquelle il exige ce traitement.

- le même mécanisme d'accord définitif est d'application.

En cas d'erreur de syntaxe, l'utilisateur est averti au moyen d'un message d'erreur, doit retourner dans l'éditeur pour corriger le fichier avant de relancer le programme.

Un message d'erreur de syntaxe est toujours accompagné de la localisation approximative de l'erreur dans le fichier de données, pointée par une flèche.

En cas d'erreur de type différent (entrées/sorties, capacité mémoire, ...), un message d'erreur seul est affiché.

1.9 EXPLICITATION DES MESSAGES D'ERREUR

01. *nom invalide*

Erreur produite lorsqu'un nom ne commence pas par une lettre ou est absent.

02. *nom de paragraphe apparaissant deux fois*

03. *nom de paragraphe attendu*

Erreur produite lorsqu'un nom de paragraphe est erroné ou que son séparateur est erroné.

06. *bornes menant à un intervalle vide*

Erreur produite lorsque la borne inférieure d'une famille définie par intervalle est supérieure ou égale à sa borne supérieure.

07. *borne supérieure invalide*

Erreur produite lorsque la borne supérieure n'est pas un entier ou est absente.

08. *borne inférieure invalide*

Erreur produite lorsque la borne inférieure n'est pas un entier ou est absente.

09. *"_" attendu*

10. *"]" attendu*

11. *énumération trop longue*

Erreur produite lorsque le nombre de valeurs introduites dans une constante est supérieur à celui attendu en fonction des noms de familles y intervenant.

12. *")" attendu*

14. *"=" attendu*

15. *"(" ou "[" attendu*

16. *";" attendu*

19. *"," ou ")" attendu*

20. *nom de famille invalide*

Erreur produite lors de l'apparition dans une constante ou une variable d'un nom de famille qui n'a pas été déclaré.

21. *trop d'indices*

Erreur produite lorsque le nombre de noms de familles dans les parenthèses d'une constante ou d'une variable ou dans les "#" excède le maximum permis, actuellement fixé à 3.

22. *"," attendu*

23. *"(" ou "=" attendu*

24. *"#" attendu*

26. *déclaration incohérente*

Erreur produite en cas d'incohérence dans les noms de famille d'une constante ou d'une variable n'apparaissant pas pour la première fois ou en cas d'incohérence facilement détectable entre noms de familles d'une constante ou d'une variable et noms de familles de quantification universelle.

27. *type de valeur invalide*

Erreur produite lorsque le type de la valeur introduite ne correspond pas à celui attendu.

29. *énumération insuffisante*

Erreur produite lorsque le nombre de valeurs introduites dans une constante est inférieur à celui attendu en fonction des noms de familles y intervenant.

31. *trop de chaînes alphanumériques*

Erreur produite lorsque le nombre total de chaînes apparaissant dans toutes les familles définies par énumération excède le maximum autorisé, actuellement fixé à 200.

32. *trop de réels à mémoriser*

Erreur produite lorsque le nombre total de valeurs réelles introduites au niveau des constantes dépasse le maximum autorisé, actuellement fixé à 4500.

34. *paragraphe "equ :" attendu*

35. *ordre des paragraphes incohérent*

37. *nom déjà utilisé*

38. *"[" attendu*

39. *valeur non comprise dans les bornes*

Erreur produite lorsqu'a été indiquée pour une famille définie par intervalle une valeur particulière non comprise dans l'intervalle de définition.

41. *nom non repris dans l'énumération*

Erreur produite lorsqu'a été indiquée pour une famille définie par énumération une chaîne particulière n'appartenant pas à l'énumération.

42. *mot réservé*

Erreur produite lorsqu'un mot réservé a été indûment utilisé.

43. *"min" ou "max" attendu*

Erreur produite lorsque la fonction objectif n'est pas introduite par l'un de ces termes ou lorsqu'elle est absente.

44. *trop de noms de contraintes*

Erreur produite lorsque le nombre de noms de contraintes dépasse le maximum permis, actuellement fixé à 30.

46. *trop d'éléments de somme*

Erreur produite lorsque le nombre d'éléments de somme dépasse le maximum autorisé, actuellement fixé à 7.

48. *"(" attendu*

49. *"=", "<=" ou ">=" attendu*

50. *variable hors position*

Erreur produite lorsqu'une variable apparaît dans le membre de droite d'une contrainte.

51. *élément de somme invalide*

Erreur produite lorsque l'élément de somme contient une multiplication par 0.

53. *">" attendu*

54. *décalage impossible*

Erreur produite lorsque la famille à droite du symbole "><" n'est pas définie par intervalle ou que son intervalle n'a pas la même largeur que celui de la famille apparaissant à gauche du "><".

56. *entier ou "_" attendu*

57. *entier ou "[" attendu*

58. *trop de noms de familles*

Erreur produite lorsque le nombre total de noms de familles déclarés dépasse le maximum autorisé, actuellement fixé à 15.

59. *trop de noms de constantes*

Erreur produite lorsque le nombre total de noms de constantes déclarés dépasse le maximum autorisé, actuellement fixé à 15.

60. *valeur ou nom de famille attendu*

62. *constante hors position*

Erreur produite lorsqu'une constante apparaît dans le membre de gauche d'une contrainte sans être multipliée par une variable.

63. *réel hors position*

Erreur produite lorsqu'un réel apparaît dans le membre de gauche d'une contrainte sans être multiplié par une variable.

64. *famille apparaissant dans "#" et dans "som"*

Erreur caractérisant une incohérence particulière.

65. *famille superflue*

Erreur produite lorsqu'apparaît dans un élément de somme un nom de famille qui ne doit pas s'y trouver, essentiellement si cette famille apparaît dans les familles de sommation mais pas dans celles de constante ni de variable ou si elle apparait dans les familles de sommation alors que dans les familles de constante et de variable elle est suivie du symbole "><" et d'une valeur particulière ou d'une chaîne particulière.

66. *famille manquante*

Erreur produite lorsque n'apparaît pas dans les familles de sommation d'un élément de somme une famille qui devrait s'y trouver, compte tenu de celles apparaissant dans la constante et/ou la variable.

67. *fin de fichier inattendue*

Erreur produite essentiellement losque "fin :" a été oublié.

68. *trop de noms de variables*

Erreur produite lorsque le nombre de noms de variables introduits dépasse le maximum autorisé, actuellement fixé à 20.

70. *indice résultant hors bornes*

Erreur produite lorsque, à cause d'une manipulation telle que le décalage, la valeur prise par un indice n'est pas comprise dans ses bornes.

71. *création de structures dépassant la capacité mémoire*

Erreur produite lorsque le dernier des enregistrements crées dynamiquement par le paramétrage conduit à un dépassement de capacité.

72. *tableau du simplexe trop grand*

Erreur produite lorsque la taille du problème à traiter conduit à un tableau du simplexe dont les dimensions excèdent les valeurs actuelles, fixées à 50 lignes et 90 colonnes.

73. *impossible de créer le fichier de sortie*

Erreur produite lorsqu'il n'y a plus de place dans la directory ou lorsque le chemin d'accès n'est pas valable.

74. *problème non traitable*

Erreur produite lorsque l'on ne peut résoudre le problème pour d'autres raisons que celles reprises sous les autres messages dont le numéro est supérieur ou égal à 70.

75. *les contraintes sont incompatibles*

76. *pas de solution réalisable*

77. *solutions optimales non bornées*

CHAPITRE 2 : CONSTRUCTION D'UNE REPRESENTATION EN MEMOIRE

2. CONSTRUCTION D'UNE REPRESENTATION EN MEMOIRE

Nous avons choisi comme structure privilégiée de représentation des données en mémoire le tableau d'enregistrements. En effet, même si une telle structure peut conduire à une consommation importante d'espace mémoire par rapport à la place nécessitée par les données effectivement présentes, elle offre une plus grande facilité de manipulation des objets et une meilleure lisibilité du programme qu'une gestion systématique de pointeurs.

Passons en revue l'application de ce principe général à chacun des trois premiers paragraphes de déclarations.

2.1 FAMILLES D'INDICES

A partir d'une déclaration de famille dont la syntaxe est correcte, le logiciel construit un enregistrement comportant quatre champs :

1. un champ booléen dont la valeur permet de savoir si la famille a été déclarée par intervalle ou énumération.

2. une chaîne de caractères donnant le nom de la famille.

3 et 4. deux champs entiers indiquant respectivement les bornes inférieure et supérieure de l'intervalle d'une famille déclarée par intervalle ou respectivement les indices de début et de fin de la

mémorisation des valeurs des noms intervenant dans l'énumération dans un vecteur auxiliaire de chaînes de caractères dévolu uniquement à cette fonction.

Exemple :

```
fam : I = [1_20];
      ville = (Liège, Namur, Bruxelles);
```

Ces déclarations permettent la construction de la représentation suivante :

true	I	1	20
false	ville	x	y

Liège	Namur	Bruxelles
^		^
x		y

2.2 CONSTANTES

Nous avons choisi de limiter à trois le nombre maximal d'indices présents dans les parenthèses des constantes, variables et du symbole "Som". En effet, un nombre appréciable de problèmes sont couverts en-deçà de cette limite.

A partir d'une déclaration de constante dont la syntaxe est correcte et d'une paragraphe de familles correctement déclaré et traité, le logiciel produit un enregistrement comportant cinq champs :

1. une chaîne de caractères donnant le nom de la constante.

2 et 3. deux vecteurs entiers de dimension 3 (nombre maximal d'indices) permettant de retrouver parmi les familles reprises dans le tableau constitué lors du traitement des déclarations du paragraphe précédent celles qui apparaissent respectivement derrière d'éventuels quantificateurs universels et dans d'éventuelles parenthèses de la constante.

4 et 5. deux champs entiers indiquant respectivement l'indice de début de la mémorisation des valeurs de constantes dans un vecteur de réels auxiliaire remplissant uniquement cette fonction et le nombre total de valeurs enregistrées dans ce tableau.

Exemple :

Supposons qu'à partir des déclarations de familles le logiciel a produit le tableau d'enregistrements suivant :

true	I	1	2
true	J	1	3
true	K	1	4

Les déclarations de constantes sont :

```
cst : Scalaire = 2;
      #I, Vecteur(I) = 1;
      Matrice(I, J) = 3, 4, 5, 6, 7, 8;
      #I, #K, Tab(I, J, K) = 9, 8, 7;
```

La représentation construite à partir de ces déclarations est la suivante :

Scalaire	0 0 0	0 0 0	x	1
Vecteur	1 0 0	1 0 0	y	2
Matrice	1 2 0	1 2 0	z	6
Tab	1 3 0	1 2 3	t	24

2 1 1 3 4 5 6 7 8
^ ^ ^
x y z

9 9 9 9 8 8 8 8 7 7 7 7
^
t

9 9 9 9 8 8 8 8 7 7 7 7

Remarques :

- il est rappelé que l'enregistrement des constantes s'effectue en considérant que l'indice le plus à droite varie le plus rapidement.
- l'enregistrement est explicite.

2.3 FONCTION OBJECTIF ET CONTRAINTES

Il nous faut distinguer ici la représentation construite pour la fonction objectif de celle produite pour une contrainte.

2.3.1. fonction objectif

A partir d'une déclaration correcte de la fonction objectif et de paragraphes de familles et de constantes correctement déclarés et traités, la représentation construite par le logiciel comprend onze champs :

1. un champ booléen permettant de déterminer la nature de l'optimisation demandée, maximisation ou minimisation.

2. une matrice comportant 3 colonnes et autant de lignes que le nombre maximal d'éléments de somme permis , permettant de retrouver pour chaque élément de somme susceptible d'apparaître dans la fonction objectif les noms des familles intervenant dans les parenthèses qui suivent obligatoirement un symbole "Som" éventuel.

3. deux matrices de même type que celle définie au point 2, reprenant pour chaque famille apparaissant dans les parenthèses du symbole "Som" éventuel :

a. un code indiquant si elle est ou non suivie d'un symbole "<>" et si c'est le cas la nature de ce qui suit ce symbole "<>" (pour rappel, valeur entière appartenant à un intervalle ou valeur apparue au sein d'une énumération).

b. la valeur entière ou le lieu de mémorisation de la valeur énumérée, selon ce qui suit effectivement le symbole "<>".

4 et 5. deux vecteurs entiers de dimension égale au nombre maximal d'éléments de somme permis et permettant de retrouver pour chacun d'eux respectivement la constante et/ou la variable y intervenant.

6 et 7. deux fois deux matrices de même type que celle définie au point 2, les deux premières se rapportant aux éventuelles familles apparaissant dans la constante, les deux autres se rapportant à celles suivant éventuellement la variable. Leur rôle respectif, pour chaque famille :

a. un code indiquant si elle est ou non suivie du symbole d'égalité généralisée ">><" et si c'est le cas la nature de ce qui suit ce symbole ">><" (pour rappel, valeur entière appartenant à un intervalle ou valeur apparue au sein d'une énumération ou nom de famille pour évolution en parallèle).

b. la valeur entière, le lieu de mémorisation de la valeur énumérée ou la référence à la famille, selon ce qui suit effectivement le symbole ">><".

8. un vecteur réel, de même dimension que ceux du point 4 et 5, destiné à recevoir la valeur du facteur de multiplication intervenant dans chaque élément de somme.

2.3.2 contrainte

A partir d'une déclaration de contrainte correcte et de paragraphes de familles et de constantes correctement déclarés et traités, une représentation analogue à celle produite pour la fonction objectif est construite. Notons les différences suivantes :

1. pas de champ booléen.

2. les structures des champs 2 à 8 du point ci-dessus sont reprises telles quelles mais dédoublées, chaque groupe représentant un membre de la contrainte, le second membre ne reprenant cependant pas d'informations relatives aux variables.

3. viennent s'ajouter :

. une chaîne de caractères donnant le nom de la contrainte.

. un vecteur de dimension 3 permettant de retrouver les familles intervenant derrière d'éventuels quantificateurs universels.

. associés à ce vecteur, deux autres de même dimension reprenant pour chaque famille apparaissant derrière des quantificateurs universels les mêmes informations que celles explicitées au point 3 de la représentation des éléments de la fonction objectif.

. un champ entier qui sera le reflet codé du séparateur ou du groupe de séparateurs intervenant entre les deux membres de la contrainte.

Soit un total de vingt-deux (10 + 7 + 5) champs.

Remarque importante : la construction d'un tableau d'enregistrements relatif aux variables du problème s'effectue en parallèle avec le traitement des déclarations de fonction objectif et de contraintes, la première apparition d'une variable et de ses éventuels indices tenant lieu de déclaration pour cette variable. L'enregistrement relatif à une variable comporte deux champs :

1. une chaîne de caractères donnant le nom de la variable.

2. un vecteur entier de dimension 3 permettant de retrouver les familles intervenant dans les éventuelles parenthèses de la variable.

Exemples :

1. Avec les déclarations suivantes,

```
fam : I = [1_2];
      J = [1_3];
cst : Scalaire = 30;
      Vecteur(I) = 10, 20;
      #I, Tab1(I) = 40;
      Matrice(I, J) = 1, 2, 3, 4, 5, 6;
      #J, Tab2(I, J) = 7, 8;
equ : Max [ -3 * Som(I) Vecteur(I) * Var1(I) +
          + 2 * Scalaire + 2 * Som(I, J) 4 *
          Matrice(I, J) * X(I, J)];
      Contr1, #J, Som(I) Vecteur(I) X(I, J) <=
                  Som(I) Tab2(I, J);
      Contr2, #I, Vecteur(I) * Var1(I) + Var2 +
                  Som(J) Matrice(I, J) * X(I,J)
                <= Tab1(I) + Som(J) Tab2(I, J);
fin :
```

le logiciel construit les représentations :

```
true      I      1      2
true      J      1      3
```

Scalaire	0 0 0	0 0 0	x	1
Vecteur	1 0 0	1 0 0	y	2
Tab1	1 0 0	1 0 0	z	2
Matrice	0 0 0	1 2 0	t	6
Tab2	2 0 0	1 2 0	a	6

```
30    10    20    40    40    1     2     3     4     5     6
^     ^           ^           ^
x     y           z           t

                  7     7     7     8     8     8
                  ^
                  a
```

Pendant la lecture du paragraphe "Equ", des représentations sont produites en parallèle pour la fonction objectif, les contraintes et les variables :

fonction objectif

```
false                                   } booléen

1 0 0       0 0 0       0 0 0           } familles d'indices
                                        } de sommation et
0 0 0       0 0 0       0 0 0           } leurs éventuelles
                                        } particularités
1 2 0       0 0 0       0 0 0           }

2           0 0 0       0 0 0           } références aux
                                        } constantes et
1           0 0 0       0 0 0           } particularités
                                        } éventuelles des
4           0 0 0       0 0 0           } familles d'indices

1           0 0 0       0 0 0           } références aux
                                        } variables et
0           0 0 0       0 0 0           } particularités
                                        } éventuelles des
2           0 0 0       0 0 0           } familles d'indices

-3                                      }
                                        } facteurs de
2                                       } multiplication
                                        }
8                                       }
```

variables

```
Var1        1 0 0

X           1 2 0
```

contraintes

```
Contr1                                  } nom

                                        } familles d'indices
                                        } de quantification
2 0 0     0 0 0     0 0 0               } universelle et leurs
                                        } éventuelles
                                        } particularités

3                                       } code du séparateur

premier membre :

                                        } familles d'indices
1 0 0     0 0 0     0 0 0               } de sommation et
                                        } leurs éventuelles
                                        } particularités

                                        } référence à la
                                        } constante et
2         0 0 0     0 0 0               } particularités
                                        } éventuelles des
                                        } familles d'indices

                                        } référence à la
                                        } variable et
2         0 0 0     0 0 0               } particularités
                                        } éventuelles des
                                        } familles d'indices

1                                       } facteur de
                                        } multiplication

second membre  :

1 0 0     0 0 0     0 0 0

5         0 0 0     0 0 0

1


Contr2

1 0 0     0 0 0     0 0 0

3
```

premier membre :

```
0 0 0     0 0 0     0 0 0
0 0 0     0 0 0     0 0 0
2 0 0     0 0 0     0 0 0

2         0 0 0     0 0 0
0         0 0 0     0 0 0
4         0 0 0     0 0 0

1         0 0 0     0 0 0
3         0 0 0     0 0 0
2         0 0 0     0 0 0

1
1
1
```

second membre :

```
0 0 0     0 0 0     0 0 0
2 0 0     0 0 0     0 0 0

3         0 0 0     0 0 0
5         0 0 0     0 0 0

1
1
```

variables

ajout de l'enregistrement :

Var2 0 0 0

2. Considérons les déclarations :

```
equ : Min [3 * X - 2.5 * Y + 3 * Z];
      Un, X - 3 * Y >= 10;
      Deux, -4 * Y + 2.5 * Z >= 8.5
fin :
```

Représentations construites :

fonction objectif

true

```
0 0 0     0 0 0     0 0 0
0 0 0     0 0 0     0 0 0
0 0 0     0 0 0     0 0 0

0         0 0 0     0 0 0
0         0 0 0     0 0 0
0         0 0 0     0 0 0

1         0 0 0     0 0 0
2         0 0 0     0 0 0
3         0 0 0     0 0 0

3
-2.5
3
```

variables

```
X       0 0 0
Y       0 0 0
Z       0 0 0
```

contraintes

Un

0 0 0 0 0 0 0 0 0

2

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

1 0 0 0 0 0 0

2 0 0 0 0 0 0

1

-3

second membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

10

Deux

0 0 0 0 0 0 0 0 0

2

premier membre :

```
0 0 0     0 0 0     0 0 0
0 0 0     0 0 0     0 0 0

0         0 0 0     0 0 0
0         0 0 0     0 0 0

2         0 0 0     0 0 0
3         0 0 0     0 0 0

-4
2.5
```

second membre :

```
0 0 0     0 0 0     0 0 0

0         0 0 0     0 0 0

8.5
```

2.4 EXEMPLE

Afin d'illustrer l'utilité des mécanismes introduits dans les conventions préliminaires 5 à 6b, nous présentons ici les structures construites pour l'exemple exposé dans l'introduction et dont une structure syntaxique correcte a été donnée à la fin du chapitre précédent.

Représentation des familles d'indices :

true	i	1	10
true	j	1	3
true	k	1	5
true	t	1	12
true	tp	0	11

Représentation des constantes et de leurs valeurs :

a	4 0 0	2 4 0	xa	36
b	4 3 0	3 4 0	xb	60
c	2 4 0	2 3 4	xc	180
d	4 0 0	1 4 0	xd	120
h	1 0 0	1 4 0	xh	120
init	1 0 0	1 0 0	xi	10
fin	1 0 0	1 0 0	xf	10
p	4 0 0	1 4 0	xp	120
s	1 4 0	1 4 0	xs	120

```
{3}12 {4}12 {5}12 {4}12 {4}12 {3}12 {4}12 {4}12
 ^                 ^          ^
xa                xb        b(3, t)

   { {3.2}12 {4.5}12 {6}12 {2.1}12 {4}12 }3
      ^
     xc

{3}12 {2}12 {3}12 2 2 2 4 2 2 2 2 2 2 2 2 {4}12 {2}12
 ^                        ^
xd                      d(4, 4)

{3}12 {3}12 {4}12 {4}12 {1 2 3 0 5 6 7 8 9 10 11 12}10
                          ^     ^
                         xh   h(i, 4)

0 0 1 0 3 0 2 0 0 0 {1}10 {1}12 {2}12 {3}12 {4}12 {5}12
^   ^   ^   ^       ^     ^
xi init(3) init(7)  xf    xp
     init(5)

      {6}12 {7}12 {8}12 {9}12 {10}12 {0.85}120
                                       ^
                                      xs
```

Représentation de la fonction objectif et des contraintes, construction du tableau des variables :

fonction objectif

false

1 4 0 0 0 0 0 0 0

2 3 4 0 0 0 0 0 0

1 0 0 0 0 0 0 0 0

1 4 0 0 1 0 0 1 0

code indiquant que le nom de famille est suivi d'un entier — cet entier

1 4 0 0 0 0 0 0 0

8 0 0 0 0 0 0

3 0 0 0 0 0 0

5 0 1 0 0 1 0

comme ci-dessus

5 0 0 0 0 0 0

5 0 0 0 0 0 0

1 0 0 0 0 0 0

2 0 0 0 0 0 0

3 0 0 0 0 0 0

4 0 3 0 0 5 0

code indiquant que le nom de famille est suivi d'un autre nom de famille — la référence à cette autre famille

4 0 0 0 0 0 0

1

-1

-0.5

-0.5

-1

variables

E 1 4 0

W 2 3 4

varinit 1 0 0

II 1 4 0

contraintes

prod1

2 4 0 0 0 0 0 0 0

1

premier membre :

3 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

2 0 0 0 0 0 0

5 0 0 0 0 0 0

1

-1

second membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0

variables

ajout en cinquième position de l'enregistrement :

V 2 4 0

prod2

2 4 0 0 0 0 0 0 0

1

premier membre :

1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

6 0 0 0 0 0 0

5 0 0 0 0 0 0

1

-1

second membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0

variables

ajout en sixième position de l'enregistrement :

U 1 4 0

dispo1

3 4 0 0 0 0 0 0 0

3

premier membre :

2 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

2 0 0 0 0 0 0 0

1

second membre :

0 0 0 0 0 0 0 0 0

2 0 0 0 0 0 0 0

1

dispo2

2 4 0 0 0 0 0 0 0

3

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

5 0 0 0 0 0 0

1

second membre :

0 0 0 0 0 0 0 0 0

1 0 0 0 0 0 0

1

equi1

1 0 0 0 0 0 0 0 0

1

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

4 0 1 0 0 1 0

6 0 1 0 0 1 0

1 0 1 0 0 1 0

1

-1

1

second membre :

0 0 0 0 0 0 0 0 0

6 0 0 0 0 0 0

1

equi2

1 4 0 0 1 0 0 1 0

1

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 0 0 0 0 0 0

4 0 3 0 0 5 0

6 0 0 0 0 0 0

1 0 0 0 0 0 0

1

-1

-1

1

second membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

0

final

1 0 0 0 0 0 0 0 0

1

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0

4 0 1 0 0 12 0

1

second membre :

0 0 0 0 0 0 0 0 0

7 0 0 0 0 0 0

1

initial

1 0 0 0 0 0 0 0 0

1

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

3 0 0 0 0 0 0

1

second membre :

0 0 0 0 0 0 0 0 0

6 0 0 0 0 0 0

1

fraction

1 4 0 0 0 0 0 0 0

1

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

7 0 0 0 0 0 0

1

second membre :

0 0 0 0 0 0 0 0 0

9 0 0 0 0 0 0

1

variables

ajout en septième position de l'enregistrement :

vars 1 4 0

vente1

1 4 0 0 0 0 0 0 0

3

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

4 0 0 0 0 0 0

1 0 0 0 0 0 0

7 0 0 0 0 0 0

1

-1

second membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

0

vente2

1 4 0 0 0 0 0 0 0

3

premier membre :

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0

1 0 0 0 0 0 0

1

second membre :

0 0 0 0 0 0 0 0 0

4 0 0 0 0 0 0

1

CHAPITRE 3 : SIMPLEXE

3. SIMPLEXE

Après un bref rappel théorique, nous nous attacherons à décrire ici les différents points ayant fait l'objet d'une implémentation.

3.1 RAPPEL THEORIQUE

3.1.1 Méthode du simplexe

Nous distinguons ici *algorithme* et *méthode* du simplexe. L'*algorithme* du simplexe, applicable aux problèmes où l'optimisation est une minimisation, où toutes les contraintes sont des égalités, où toutes les variables sont positives et où l'on connaît une base réalisable de départ, est destiné à l'obtention de la solution proprement dite : valeur de la fonction objectif, variables en base et leurs valeurs. La *méthode* du simplexe se propose de lever l'hypothèse portant sur la connaissance d'une base réalisable de départ, permet de déterminer si les contraintes sont compatibles, si le système des contraintes contient des équations redondantes et si la fonction objectif possède un optimum fini sous ces contraintes. Enfin, cette méthode contient l'algorithme du simplexe comme technique de calcul. Il existe en fait deux algorithmes de résolution appelés respectivement *algorithme primal* et *algorithme dual*.

3.1.2 Analyse de sensibilité

Ce problème consiste en la détermination d'un intervalle de variation d'une ou plusieurs données pour lequel la base obtenue reste optimale, ce qui permet de mesurer la sensibilité de la solution optimale à des changements d'hypothèses.

3.1.3 Paramétrage

Il s'agit ici de faire varier certaines données de façon continue et d'étudier le comportement de la solution optimale en fonction des valeurs variables de ces données. Ce problème est en fait une extension du précédent.

3.2 POINTS D'IMPLEMENTATION

3.2.1 Méthode du simplexe

L'algorithme primal du simplexe a été implémenté dans sa forme traditionnelle, aucune technique de calcul particulière n'ayant été utilisée.

Le risque de cyclage étant mineur, aucune méthode n'a été implémentée pour l'éviter. Il est à noter que dans le cadre restreint de ce problème précis, les erreurs d'arrondis de la machine contribuent sensiblement à réduire les risques.

L'implémentation de la méthode des deux phases permet de lever l'hypothèse portant sur la connaissance d'une base réalisable de départ.

Enfin, l'algorithme dual a été également implémenté de façon à ne pas devoir systématiquement faire appel à la méthode des deux phases.

Le programme prend en charge l'ajout d'éventuelles variables d'écart et/ou artificielles et s'oriente automatiquement vers l'algorithme le mieux adapté au problème soumis par l'utilisateur.

3.2.2 Analyse de sensibilité

L'analyse de sensibilité portera soit sur l'un des coûts de la fonction objectif, soit sur le second membre d'une seule contrainte.

3.3.2 Paramétrage

Les possibilités offertes ici sont celles où soit l'un des coûts de la fonction objectif, soit le second membre d'une seule contrainte dépend linéairement d'un seul paramètre.

CHAPITRE 4 : TRANSFORMATION DE STRUCTURES

4. TRANSFORMATION DE STRUCTURES

Après avoir exposé les structures générées par la reconnaissance de la syntaxe, nous rappelons brièvement ici les structures sur lesquelles repose généralement la méthode du simplexe ainsi que l'analyse de sensibilité et le paramétrage. Nous envisagerons plus particulièrement les structures nécessaires à l'implémentation des points décrits au chapitre précédent et dirons quelques mots de la création de ces dernières à partir des constructions effectuées par la reconnaissance de la syntaxe.

4.1 METHODE DU SIMPLEXE

Les structures reprises ici sont traditionnelles :

- un vecteur réel pour les coûts, coéfficients de la fonction objectif.
- un vecteur réel pour les seconds membres des contraintes.
- un réel pour exprimer la valeur de la fonction objectif.
- une matrice destinée à la représentation des premiers membres des contraintes.

L'implémentation de la méthode des deux phases exige l'ajout des structures suivantes :

- un vecteur réel pour les coûts du problème auxiliaire.

- un réel pour exprimer la valeur de la fonction objectif du problème auxiliaire.

L'implémentation de l'algorithme dual ne demande aucune structure supplémentaire.

4.2 ANALYSE DE SENSIBILITE

Cette étude nécessite l'ajout des structures suivantes :

- un vecteur réel pour la variation des coéfficients de la fonction objectif.

- un vecteur réel pour la variation des seconds membres des contraintes.

_ un réel pour exprimer la variation de valeur de la fonction objectif.

4.3 PARAMETRAGE

Ce traitement ne nécessite en lui-même aucune structure additionnelle. Cependant, pour des raisons de convivialité, une visualisation claire des résultats étant souhaitée, une structure d'enregistrement a été construite. Elle contient :

- un réel exprimant la valeur de la fonction objectif, partie indépendante du paramètre.

- un réel exprimant la valeur de la fonction objectif, partie dépendante du paramètre.

- un vecteur entier contenant les numéros de colonne des variables en base.

- un vecteur réel contenant la valeur de chacune de ces variables.

- deux réels donnant l'intervalle dans lequel la solution est valable.

- un pointeur vers l'enregistrement suivant.

Le paramétrage consistant en une exploration à gauche et à droite de la solution de départ, il nous faut, après avoir effectué un ensemble de pivotages d'un côté, revenir à la solution de départ pour développer l'analyse de l'autre côté. En conséquence, des structures sont nécessaires pour mémoriser cette solution dans son ensemble, à savoir :

- une matrice représentant l'état des premiers membres des contraintes.

- un vecteur réel pour les coéfficients de la fonction objectif, partie indépendante du paramètre.

- un vecteur réel pour les coéfficients de la fonction objectif, partie dépendante du paramètre.

- un vecteur réel pour les seconds membres des contraintes, partie indépendante du paramètre.

- un vecteur réel pour les seconds membres des contraintes, partie dépendante du paramètre.

- un réel exprimant la valeur de la fonction objectif, partie indépendante du paramètre.

- un réel exprimant la valeur de la fonction objectif, partie dépendante du paramètre.

En outre, la nature même du traitement des problèmes d'analyse de sensibilité et de paramétrage exige la mémorisation de certaines caractéristiques des solutions, à savoir :

- un vecteur entier contenant les numéros des colonnes où l'on peut retrouver l'inverse de la base.

- un vecteur entier contenant les numéros de colonne où l'on peut trouver la base dans l'état actuel du traitement.

- un vecteur réel contenant les coûts initiaux des variables en base dans l'état actuel du traitement.

Enfin, des considérations techniques conduisent bien sûr à la création de structures de travail pour la description desquelles nous renvoyons le lecteur à l'appendice 7.1.

4.4 TRANSFORMATION

Ce sujet bien particulier est difficile à aborder sans entrer dans des considérations techniques qui, finalement, ne sont pas essentielles. Nous n'en dirons donc que quelques mots.

Il s'agit bien sûr d'initialiser correctement les vecteurs des coûts et des seconds membres des contraintes

ainsi que la matrice des premiers membres des contraintes, à partir des structures obtenues par la reconnaissance de la syntaxe et de la méthode employée pour la résolution du problème.

En effet, en fonction de la méthode de résolution choisie par le programme, des manipulations de dernière minute, un changement de signe, par exemple, peuvent être faites dans les équations. Ces manipulations, tout comme l'éventuel ajout ultérieur de variables artificielles et/ou d'écart, sont transparentes pour l'utilisateur qui retrouvera toujours son fichier de données intact. Ce sont les structures éventuellement manipulées qui sont à transformer, à "traduire".

La difficulté majeure à résoudre ici est en fait la traduction correcte des structures rendant compte des notations du type "<>" et surtout "><". Ces cas particuliers font l'objet d'un traitement spécial pour la description duquel nous renvoyons le lecteur à l'appendice 7.1.

CHAPITRE 5 : IMPLEMENTATION

5. IMPLEMENTATION

Nous traitons dans ce chapitre des divers problèmes concrets rencontrés lors de l'implémentation et des solutions qui leur ont été apportées.

5.1 PROBLEMES RELATIFS A LA PLACE MEMOIRE

Le problème posé ici résulte en fait aussi bien de l'ordinateur sur lequel l'implémentation a été réalisée (IBM PC) que du langage de programmation utilisé (Turbo Pascal). Par conséquent, afin de permettre une compréhension approfondie de ce problème, il nous faut tout d'abord rappeler d'une part quelques caractéristiques essentielles du microprocesseur 8086 qui équipe l'IBM PC et d'autre part leurs conséquences au niveau du Turbo Pascal.

5.1.1 Microprocesseur 8086

Le microprocesseur 8086 a un espace d'adressage de 1 méga-octet. Chaque adresse est donc définie par un mot de 16 bits. Pour gérer ces adresses, le microprocesseur utilise le format "segment : déplacement", ce qui signifie que toute adresse s'exprime sous forme de numéro de segment et de déplacement dans ce segment. Un segment est en fait une zone de 64 K octets dans l'espace d'adressage. Quatre des seize registres du 8086 sont utilisés comme registres de segments : registre de segment de code, registre de segment de données, registre de segment de pile, registre de segment supplémentaire.

5.1.2 Turbo Pascal

Globalement, le Turbo Pascal répartit sa charge de la manière suivante : le code est limité au segment de code, les variables globales au segment de données tandis qu'une gestion adéquate est prévue pour que les variables pointées et les variables locales puissent déborder du segment de pile dans une certaine mesure. En l'absence de toute documentation à ce sujet, il semble cependant que malgré ce possible débordement, les variables locales à une même procédure ne peuvent occuper plus de 64 K octets en mémoire.

5.1.3 Conséquences

La structure et certaines données du programme ont été bien évidemment influencées par les limites strictes de disponibilité mémoire relatives aux variables globales et au code ainsi que par celles, plus larges, imposées à la pile.

En particulier, il a été décidé de faire usage du système "overlay" qui permet de rassembler un certain nombre de sous-programmes (procédures et/ou fonctions) dans un fichier séparé de celui contenant le programme "principal" (variables globales, procédures et/ou fonctions qui doivent être appelables à partir de toutes les autres, code du programme principal proprement dit), sous-programmes qui pourront être chargés *un à la fois* dans une *même zone* de mémoire. Ce principe doit être appliqué avec précaution afin de ne pas affecter les performances. Il faut en outre choisir soigneusement les

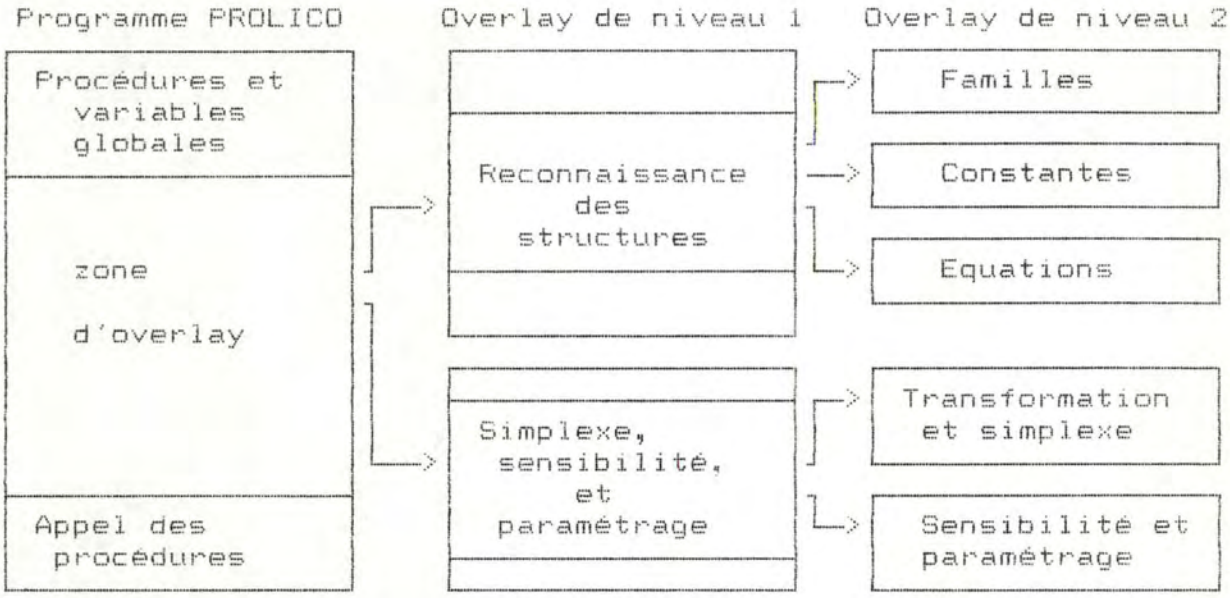
procédures et fonctions à rassembler et veiller en particulier à ce qu'elles ne s'appellent pas mutuellement puisqu'elles ne seront présentes en mémoire que tour à tour. Dans le cas présent, le découpage effectué au sein du programme, se justifiant essentiellement par le fait que le code lui-même dépasse 64 K octets, s'efforce de correspondre à un découpage logique guidé par les actions à effectuer : reconnaissance de la syntaxe, subdivisée en fonction des paragraphes à traiter (familles, constantes, fonction objectif et contraintes, commentaires) et calculs, ceux-ci se subdivisant en transformation de structures et simplexe d'une part, module d'analyse (sensibilité et paramétrage) d'autre part. Il peut être schématisé de la façon suivante :

- un premier niveau d'"overlay" permet de considérer successivement dans la même zone de mémoire la reconnaissance de la syntaxe puis les calculs.

- au sein de la reconnaissance de la syntaxe, un second niveau d'"overlay" envisagera selon leur apparition le traitement des différents paragraphes.

- au sein des calculs, un second niveau d'"overlay" verra d'abord, groupés, la transformation de structures et le simplexe proprement dit, et ensuite le module d'analyse.

Schématisation :



5.1.4 Justification des choix

Les contraintes techniques exposées ci-dessus nous ont obligés à effectuer des choix en matière de fixation des tailles des vecteurs et matrices utilisés aussi bien pour la reconnaissance de la syntaxe que pour le simplexe et le module d'analyse.

La matrice la plus volumineuse est celle des premiers membres des contraintes. Elle est à mettre en relation avec le nombre maximal de valeurs de constantes mémorisables susceptibles de la remplir. Il faut également tenir compte de la relation entre lignes et colonnes de cette matrice en matière d'ajout de variables d'écart et/ou artificielles, ce qui se réduit en fait à une relation entre nombre et type de contraintes d'une part et colonnes d'autre part.

Le cas où l'influence de la relation entre lignes et colonnes est la plus importante est celui où toutes les contraintes sont de type ">=" et où l'algorithme dual ne peut être utilisé en raison du signe des coûts de la fonction objectif. Soient COL le nombre total de colonnes, LI le nombre total de lignes et VAR le nombre de colonnes nécessaires aux variables du problème proprement dit. L'hypothèse ci-dessus conduit à la formule suivante, reflétant l'ajout automatique de variables d'écart et/ou artificielles :

$$COL = VAR + 2 * LI$$

Vouloir respecter scrupuleusement cette contrainte engendrerait la création d'une matrice réelle monstrueuse

qui aurait tôt fait de dépasser la capacité mémoire, même pour un problème de taille "raisonnable", ou ne permettrait que le traitement de problème de taille très réduite.

Abordons cette difficulté sous un autre angle. On observe généralement que les problèmes de minimisation s'accompagnent d'un nombre plus important de contraintes de type ">=" et que les problèmes de maximisation font intervenir plus de contraintes de type "<=". Cette constatation pourrait être schématisée comme suit, en fonction du type d'optimisation et du type de contraintes.

minimisation	maximisation
type ">=" : 60 %	type ">=" : 10 %
type "<=" : 10 %	type "<=" : 60 %
type "=" : 30 %	type "=" : 30 %
COL = VAR + 1.6 * LI	COL = VAR + 1.1 * LI

Ce genre de formules, basée sur des observations, conduit vers un choix du type :

COL = 90, LI = 50, VAR = 10 pour la minimisation
COL = 90, LI = 50, VAR = 35 pour la maximisation

D'autres combinaisons sont bien sûr possibles. Les valeurs données ici semblent acceptables et tiennent déjà compte du souci de se soumettre aux contraintes techniques exposées plus haut, ainsi que nous le verrons dans le paragraphe suivant. Insistons cependant sur le fait que ces valeurs ne sont que le résultats de calculs

approximatifs et ne dispensent nullement l'utilisateur de la vérification préalable de l'adéquation de son problème aux capacités du logiciel.

5.1.5 Calculs

A partir de ces valeurs, des calculs précis ont dû être effectués pour éviter tout dépassement de capacité du segment de données et tout dépassement de capacité du segment de pile pour les procédures "à risques". Ces calculs portent sur la place occupée par les variables déclarées en tenant compte de leurs relations et justifient les limites imposées au niveau de la syntaxe. Ils sont développés ci-dessous dans les grandes lignes.

Sachant qu'un réel occupe 6 octets, qu'un entier en exige 2, une variable booléenne 1 et une chaîne de caractères un nombre égal à sa longueur augmenté de 1, on obtient, en prenant comme référence l'ensemble des valeurs exposées dans la définition de la syntaxe :

1. variables globales

_ tableau des familles : 26 * 15 = 390 octets

- tableau des constantes : 37 * 15 = 555 octets
- fonction objectif : 113 octets
- tableau des équations : 670 * 30 = 20 100 octets
- tableau des variables : 27 * 20 = 540 octets
- tableau de mémorisation des chaînes de caractères des familles définies par énumération : 2000 octets
- tableau de mémorisation des valeurs réelles des constantes : 27 000 octets
- variables de travail : 507 octets

Total : 51 205 octets dans le segment de données.

Ajoutons cependant encore ici des tableaux de travail qui ne sont pas globaux mais qui ont été déclarés à ce niveau pour alléger le segment de pile de la procédure "calcul" (voir ci-dessous) et qui nécessitent 10 800 octets.

2. Variables locales à "calcul"

Une place importante est occupée par la matrice des premiers membres des contraintes et sa copie (2 * 50 * 90 réels = 54 000 octets). Les vecteurs des seconds membres et des coûts ainsi que leurs copies sont plus modestes tout comme le sont les diverses variables de travail, ce qui nous donne un total de 58 140 octets occupant le segment de pile de cette procédure. Il nous reste donc la possibilité de créer dynamiquement les enregistrements contenant les résultats portant sur le paramétrage. Chacun de ces enregistrements exige une place de 438 octets, ce qui nous laisse la liberté d'en créer 16.

5.2 DISQUE VIRTUEL

L'utilisateur regrettera certainement d'avoir à limiter l'usage de ce logiciel à des problèmes de taille acceptable mais relativement réduite. Nous avons envisagé de mettre à profit l'existence du disque virtuel pour accroître les possibilités du programme dans ce domaine. Cependant, il nous a semblé que le gain en matière de taille des problèmes traitables n'était pas suffisant pour compenser une certaine dégradation des performances

ainsi qu'un alourdissement notable de la programmation elle-même. En outre, la place occupée par le disque virtuel vient en réduction de celle disponible pour le segment de pile. Dans ces conditions, il n'est pas non plus possible d'affirmer qu'un gain sensible aurait été obtenu.

CHAPITRE 6 : CONCLUSION

6. CONCLUSION

Le but de ce travail est d'offrir à l'utilisateur de programmes linéaires la possibilité d'exposer le problème à traiter d'une manière naturelle, sans avoir à étudier au préalable un langage de déclaration lourd ou fastidieux, mais tout en conservant une grande souplesse d'expression. Une attention particulière a aussi été accordée au problème de la présentation des résultats.

Si le logiciel pèche essentiellement par ses performances en matière de vitesse d'exécution et surtout par la taille relativement réduite des problèmes effectivement traitables, il a atteint son but du point de vue de la définition et de la reconnaissance d'une syntaxe proche de l'écriture mathématique, admettant la quantification universelle et le symbole de sommation, adaptant des notations mathématiques particulières. Il offre également une représentation très pratique des résultats : valeur de la fonction objectif, variables en base et leurs valeurs, coûts. Au niveau de l'analyse de sensibilité, l'intervalle de validité de la solution est fourni sous la forme mathématique courante. En outre, au niveau du paramétrage était prévue une visualisation complète de l'évolution de la fonction objectif par l'intermédiaire de la mise en évidence de tous les points de changements de base, alors que la plupart des programmes disponibles ne présentent dans ce cas qu'une visualisation par intervalles de taille fixe risquant ainsi de "perdre" un éventuel changement de base apparu au sein d'un intervalle. Rappelons cependant que l'accès

à ce module a dû être supprimé pour des raisons de fiabilité.

Le problème essentiel à résoudre reste cependant celui de la place mémoire et de l'usage qui en est fait par le microprocesseur et le langage. Des voies nouvelles devraient être explorées dans le but de trouver une solution plus satisfaisante à cette entrave purement technique.

Des extensions pourraient être greffées sur ce travail, visant à améliorer les performances, permettant une visualisation graphique des résultats, étendant la syntaxe, sa reconnaissance et les traitements à d'autres types de problèmes linéaires... L'exploration de ces diverses possibilités permettraient de compléter avantageusement le travail réalisé jusqu'ici.

En outre, la conception et la mise en oeuvre de ce projet ont été une illustration utile de diverses facettes du travail d'informaticien, allant de la définition d'une syntaxe indépendante de la machine jusqu'à la découverte et l'usage de fonctions du système, nécessitant une réflexion sur la puissance de ce système mais aussi sur ses limites. Certaines limites semblent avoir également été atteintes au niveau du compilateur Turbo Pascal, lequel présente quelques défaillances lorsqu'un trop grand nombre d'instructions conditionnelles sont imbriquées.

CHAPITRE 7 : BIBLIOGRAPHIE

7. BIBLIOGRAPHIE

[DAN 66] G. B. DANTZIG, Applications et prolongements de la programmation linéaire, Dunod, Paris, 1966.

[SIM 72] M. Simmonard, Programmation linéaire. Tome 1 : Fondements, Dunod, Paris, 1972.

[AMSL 81] Lamps User Guide, Version 1.1, Advanced Mathematical Software Limited, 1981.

[FOR] J. J. H. FORREST, Magic Lamps, An Approach to Problem Solving, CAP, Mathematical Programming Systems.

[BRO] G. G. Brown, A. M. Geoffrion, G. H. Bradley, Production and Sales Planning.

Facultés Universitaires Notre-Dame de la Paix

Institut d'Informatique

PROGRAMMATION LINEAIRE CONVIVIALE SUR MICRO-ORDINATEUR

Appendices

Promoteur : Monsieur **J-P. LECLERCQ**

Mémoire présenté en vue de l'obtention du grade de Licenciée et Maître en Informatique par **Annick GERARD**

année académique 86-87

CHAPITRE 8 : APPENDICES

8. APPENDICES

8.1 PROGRAMME, STRUCTURE, DESCRIPTION

Nous nous attacherons ici à éclaircir par une description adéquate les procédures et fonctions par le biais de commentaires dans le listing du programme. Rappelons qu'une vue globale de la structure du programme a été donnée dans le chapitre consacré à l'implémentation. On remarquera que l'accès au module de paramétrage a été interdit. Certaines procédures utilisées en phase de test, affichage des constructions générées par la reconnaissance de la syntaxe, affichage de l'ensemble des variables nécessaires au traitement du simplexe, ont simplement été mises entre commentaires de façon à pouvoir être réutilisées à tout moment. Il est bien évident que ce paragraphe est essentiellement destiné à toute personne désireuse d'améliorer ou d'étendre le logiciel obtenu.

Si la partie du programme relative au simplexe proprement dit a nécessité moins d'imagination au niveau programmation, on ne peut cependant nier que les difficultés rencontrées pour la reconnaissance de la syntaxe et les transformations de structures ont exigé à la fois réflexion et astuce.

Le point fondamental au niveau de la reconnaissance de la syntaxe est le mode de lecture du fichier de données. Celui-ci est parcouru séquentiellement, sans retour en arrière, par division, une division étant un

ensemble de caractères sans séparateur suivis d'un séparateur. Une analyse est d'abord faite sur le séparateur pour voir s'il est plausible dans la situation courante et ensuite sont étudiés les autres caractères.

La transformation de structures exige la connaissance du numéro de ligne et/ou de colonne de l'élément auquel affecter une valeur. Pour les obtenir, il est nécessaire de parcourir les différentes valeurs possibles pour les indices intervenant dans l'élément de somme en cours de "traduction", les éventuelles notations mathématiques particulières et les quantifications.

```
{*************************************************************************}
{*                                                                       *}
{*       Réalisation d'un programme linéaire offrant à l'utilisateur     *}
{*       la possibilité d'introduire les données du problème de          *}
{*       manière souple par utilisation de la quantification et du       *}
{*       symbole de sommation essentiellement                            *}
{*                                                                       *}
{*************************************************************************}

program memoire;

const
      max_alpha = 100;   (* nbre max de chaînes alphanumériques mémorisables *)
      max_reel  = 4500;  (* nbre max de valeurs de constantes mémorisables   *)
      max_indices = 3;   (* nbre max d'indices dans les constantes, variables*)
                         (* quantifications universelles et sommes           *)
      max_equ = 30;      (* nbre max de noms de contraintes mémorisables     *)
      max_som = 7;       (* nbre max d'éléments de somme dans les contraintes*)
                         (* et dans la fonction objectif                     *)
      max_fam = 15;      (* nbre max de noms de familles mémorisables        *)
      max_cst = 15;      (* nbre max de noms de constantes mémorisables      *)
      max_var = 20;      (* nbre max de noms de variables mémorisables       *)
      max_li_col = 90;   (* max du nbre max de lignes et du nbre max de      *)
                         (* colonnes du tableau du simplexe                  *)
      max_equ_var = 30;  (* max du nbre max de contraintes et du nbre max de *)
                         (* variables                                        *)
      max_li = 50;       (* nombre max de lignes et de colonnes de la        *)
      max_col = 90;      (* matrice des premiers membres des contraintes     *)

type
      table1 = array[1..max_som] of integer;
      table2 = array[1..max_som] of real;
      tableau = array[1..max_indices] of integer;
      grostab = array[1..max_som] of tableau;
      str255 = string[255];
      str80 = string[80];
      str30 = string[30];
      str20 = string[20];
      str15 = string[15];
      str3 = string[3];
      nom_equ_var = array[1..max_equ_var,1..max_li_col] of integer;
                    (* record de mémorisation d'une famille  *)
      table_index = record
                     num : boolean;
                     nom : str20;
                     borne_inf,borne_sup : integer;
                    end;
                    (* record de mémorisation d'une constante *)
      table_const = record
                     nom : str20;
                     no_ent_ptt,no_ent_par : tableau;
                     debut,total : integer;
                    end;
```

```
              (* record de mémorisation de la fonction objectif *)
    fct_obj = record
               nature : boolean;
               no_ent_som,code_som,val_som,
               code_cst,code_var,val_cst,val_var : grostab;
               cst,variable : table1;
               mult : table2;
              end;
                (* record de mémorisation d'une contrainte *)
    table_equ = record
                 code_equ : integer;
                 no_ent_ptt,code_ptt,val_ptt : tableau;
                 nom : str20;
                 no1_ent_som,no2_ent_som,
                 code1_som,val1_som,code2_som,val2_som,
                 code1_cst,val1_cst,code2_cst,val2_cst,
                 code1_var,val1_var : grostab;
                 cst1,cst2,var1 : table1;
                 mult1,mult2 : table2;
                end;
                (* record de mémorisation d'une variable *)
    table_var = record
                 nom : str20;
                 no_ent_par : tableau;
                end;
    tab_str_li =  array[1..max_li] of string[44];
    tab_str_col = array[1..max_col] of string[48];
var
              (* nom du fichier de données *)
    nom_fic_in : str30;
              (* nom du fichier de résultats éventuel *)
    nom_fic_out : str30;
              (* tableau de records des familles    *)
    tab_ind : array[1..max_fam] of table_index;
              (* tableau de records des constantes  *)
    tab_cst : array[1..max_cst] of table_const;
              (* tableau de records des contraintes *)
    tab_equ : array[1..max_equ] of table_equ;
              (* tableau de records des variables   *)
    tab_var : array[1..max_var] of table_var;
              (* fonction objectif *)
    fo : fct_obj;
                (* tableau de mémorisation des chaînes *)
                (* apparaissant dans les familles      *)
                (* définies par énumération            *)
    mem_alpha : array[1..max_alpha] of str20;
                (* tableau de mémorisation des         *)
                (* valeurs de constantes               *)
    mem_const : array[1..max_reel] of real;

    oldstr,          (* ligne précédant la ligne courante *)
    strin,           (* ligne courante                    *)
    strw : str80;    (* élément courant dans la ligne courante      *)
                     (* c-à-d ensemble de caractères non séparateurs*)
                     (* terminés par un séparateur ou 80 caractères *)
```

```
    f_in,f_out : text;    (* fichiers de données et de résultats *)
    avance,ok,coherence,  (* booléens de travail                 *)
         (* orientation sur les peripheriques sortie*)
    ecran,disque,imprimante : boolean;
         (* outils de debugging *)
    debug1,debug2 : boolean;
         (* nbre d'elements de somme resp. dans le    *)
         (* premier et second membre d'une contrainte *)
    nb1_som,nb2_som : array[1..max_equ] of integer;
    nb_cst,nb_elem_cst,nb_fam,nb_elem_fam,cpt_in,
    nb_equ,nb_var,nb_fo_som : integer;
    mem,code_mem,code_fictif,val_fictif : tableau;
    var_nom : tab_str_col;
    equ_nom : tab_str_li;    (* gain de place stack *)

procedure x(str : str20); (* outil de debugging *)
 begin
  if debug2 then writeln('╠═══ entree dans ',str,' ═══╣');
 end;

procedure y(str : str20); (* outil de debugging *)
 begin
  if debug2 then writeln('├─── sortie de   ',str,' ───┤');
 end;

procedure bip;
 begin
  write(chr(7));
 end;

(*************************************************************************)
(* fonction booléenne renvoyant true si le caractère lu  à sa demande *)
(* est 'O' ou 'o' et faux si ce caractère est 'N' ou 'n'               *)
(*************************************************************************)

function reponse_clavier : boolean;
 var c : char;
 begin
  write(' (o/n) : ');
  repeat
   read(kbd,c);
  until (c in ['o','O','n','N']);
  writeln(c);
  if (c = 'o') or (c = 'O')
   then reponse_clavier := true
   else reponse_clavier := false;
 end;
```

```
(*********************************************)
(* fonction de type string convertissant un *)
(* entier I en une string de longueur LEN   *)
(*********************************************)

function i2s(i,len : integer) : str80;
 var s : str80;
 begin
  str(i:len,s);
  i2s := s;
 end;

(*********************************************)
(* fonction de type string convertissant un *)
(* réel R de longueur L1 comprenant L2      *)
(* chiffres après le point en une string    *)
(*********************************************)

function r2s(r : real; l1,l2 : integer) : str80;
 var s : str80;
 begin
  str(r:l1:l2,s);
  r2s := s;
 end;

(*************************************************)
(* fonction de type string convertissant une    *)
(* string S de longueur quelconque en string de *)
(* longueur LONG                                 *)
(*************************************************)

function s2s(s : str80; long : integer) : str80;
 var i,l : integer;
     s2 : str80;
 begin
  l := length(s);
  if l = long
   then s2s := s
   else begin
         s2 := s;
         if l < long
          then begin
                for i := l + 1 to long do s2 := ' ' + s2;
                s2s := s2;
               end
          else begin
                s2 := copy(s2,1,long);
                s2s := s2;
               end;
        end;
 end;
```

```
(*****************************************************)
(* procédures orientant la sortie de la string TXT *)
(* en fonction des périphériques de sortie choisis *)
(*****************************************************)

procedure ecris(txt : str255);
 begin
  if ecran       then write(txt);
  if disque      then write(f_out,txt);
  if imprimante then write(lst,txt);
 end;

procedure ecrisln(txt : str255);
 begin
  ecris(txt);
  if ecran       then writeln;
  if disque      then writeln(f_out);
  if imprimante then writeln(lst);
 end;

(*****************************************************)
(* fonction booléenne renvoyant true si NOM_FIC est *)
(* un nom de fichier existant et faux sinon         *)
(*****************************************************)

function existe(nom_fic : str80) : boolean;
 var ok : boolean;
     f  : file;
 begin
  assign(f,nom_fic);
  (*$I-*)
  reset(f);
  ok := (ioresult = 0);
  (*$I+*)
  if ok then close(f);
  existe := ok;
 end;

(*********************************************************)
(* fonction de type string rendant les I caractères de  *)
(* droite de STRG ou STRG elle-même si I dépasse la     *)
(* longueur de STRG ou une string vide si I est < 1     *)
(*********************************************************)

function right(strg : str80;i : byte) : str80;
 begin
  if i > length(strg)
   then right := strg
   else if i < 1
    then right := ''
    else right := copy(strg,length(strg) - i + 1,i);
 end;
```

```
(* idem pour les caractères de gauche *)

function left(strg : str80; i : byte) : str80;
 begin
  if i > length(strg)
   then left := strg
   else if i < 1
         then left := ''
         else left := copy(strg,1,i);
 end;

(*********************************************)
(* fonction renvoyant un caractère qui est la *)
(* minuscule du caractère CAR s'il y a lieu   *)
(*********************************************)

function lowcase(car : char) : char;
 var asc : byte;
 begin
  asc := ord(car);
  if (asc > 64) and (asc < 91) then asc := asc + 32;
  lowcase := chr(asc);
 end;

(*****************************************************************************)
(* procédure recevant la string ST, en éliminant les blancs et rendant *)
(* le résultat dans ST                                                       *)
(*****************************************************************************)

procedure pack(var st : str80);
 var s2 : str80;
     i : byte;
 begin
  s2 := '';
  for i := 1 to length(st) do
   if st[i] <> ' '
    then s2 := s2 + st[i];
  st := s2;
 end;
```

```
(*************************************************************************)
(* procédure affichant TXT encadré, centré et eventuellement             *)
(* en video inverse si LOW est vrai, sur la ligne NOLI                   *)
(*************************************************************************)

procedure msg(txt : str80; noli : byte; low : boolean);
 var i,l,d : byte;
 begin
  for i := 0 to 2 do
   begin
    gotoxy(1,noli + i);
    clreol;
   end;
  l := length(txt);
  d := (78 - l) div 2;
  if low then lowvideo;
  gotoxy(d,noli);
  write('╔');
  for i := 1 to l + 2 do write('═');
  write('╗',' ':d-2);
  gotoxy(d,noli + 1);
  write('║ ');
  write(txt);
  write(' ║',' ':d-2);
  gotoxy(d,noli + 2);
  write('╚');
  for i := 1 to l + 2 do write('═');
  write('╝',' ':d-2);
  if low then normvideo;
 end;
```

```
(*********************************************************************)
(* affiche le message correspondant au no NOERR et contenu dans le *)
(* fichier de messages MSGERR                                        *)
(* sortie du programme et clôture des fichiers                       *)
(* les erreurs dont le numéro est inférieur à 70 sont des erreurs    *)
(* de syntaxe pour lesquelles sont affichées la ligne où l'erreur    *)
(* a été détectée et sa localisation approximative ainsi que la      *)
(* ligne précédente                                                  *)
(* les erreurs dont le numéro est supérieur ou égal à 70 sont des    *)
(* erreurs détectées lors des calculs ou des opérations d'entrée/    *)
(* sortie pour lesquelles seul le message est affiché                *)
(*********************************************************************)

procedure erreur(noerr : integer);
 var f_err : text;
     txt,msgerr : str80;
     i      : integer;
 begin
  window(1,5,80,25);
  clrscr;
  window(1,1,80,25);
  bip;
  gotoxy(1,5);
  write('╚');
  for i := 1 to 78 do write('=');
  writeln('╝');
  writeln;
  writeln;
  ecrisln('');
  ecrisln('');
  if noerr <> 0
   then begin
         txt := '*** erreur (' + i2s(noerr,2) + ') du type ';
         if noerr > 70
          then txt := txt + 'execution ';
         assign(f_err,'msgerr.pas');
         reset(f_err);
         i := 0;
         repeat
          i := i + 1;
          readln(f_err,msgerr);
         until eof(f_err) or (i = noerr);
         close(f_err);
         if i <> noerr
          then txt := txt + '### inconnu ###'
          else txt := txt + msgerr;
         txt := txt + ' ***';
         lowvideo;
         ecrisln(txt);
         normvideo;
         ecrisln('');
```

```
        if noerr < 70
         then begin
               ecrisln(oldstr);
               if strin = '\'
                then ecrisln('')
                else ecrisln(strin);
                for i := 1 to cpt_in - 1 do ecris(' ');
                ecrisln(chr(24));
              end;
       end;
  ecrisln('');
  close(f_in);
  writeln;
  if disque then close(f_out);
  halt;
 end;

(*****************************************************************)
(* procédure demandant à l'utilisateur de pousser sur RETURN *)
(* pour continuer et attendant que cela soit fait             *)
(*****************************************************************)

procedure attente;
 var car : char;
 begin
  if ecran
   then begin
         msg('Poussez sur <RETURN> pour continuer',16,false);
         repeat
          read(kbd,car);
         until ord(car) = 13;
         gotoxy(1,16); clreol;
         gotoxy(1,17); clreol;
         gotoxy(1,18); clreol;
        end;
 end;

(*************************************************************************)
(* procédure affichant le contenu de ST en X,Y et lisant le nouveau *)
(* contenu au clavier.                                                   *)
(*************************************************************************)

 procedure lis_string(x,y : integer; var st : str80);
  var s2 : str80;
  begin
   gotoxy(x,y);
   write(st);
   clreol;
   gotoxy(x,y);
   s2 := '';
   readln(s2);
```

```
  if s2 = '//'
   then st := ''
   else if s2 <> ''
         then st := s2;
  gotoxy(x,y);
  write(st);
  clreol;
 end;

(*************************************************************************)
(* procédure affichant 'o' ou 'n' en X,Y selon le contenu de BOOL et *)
(* lisant au clavier 'o' ou 'n', ceci déterminant la nouvelle valeur *)
(* de BOOL                                                               *)
(*************************************************************************)

procedure lis_boolean(x,y : integer; var bool : boolean);
 var o : byte;
     c : char;
 begin
  gotoxy(x,y);
  if bool
   then write('o')
   else write('n');
  write(chr(8));
  repeat
   read(kbd,c);
   o := ord(c);
   if o > 31
    then case c of
          'o','O' : begin
                     bool := true;
                     write(c,chr(8));
                    end;
          'n','N' : begin
                     bool := false;
                     write(c,chr(8));
                    end;
          else ;
         end;
  until (o = 13);
 end;
```

```
(*************************************************************************)
(* procédure permettant le choix des périphériques de sortie pour  *)
(* l'utilisateur, production du message d'erreur no 73 en cas      *)
(* d'impossibilité de création du fichier de résultats sur disque  *)
(*************************************************************************)

procedure init_options;
 var i : byte;
     fini : boolean;
     cmd : str80 absolute Cseg:$80; (* destiné à la phase de test *)
     l,p : byte;
 begin
  clrscr;
  for i := 2 to 79 do
   begin
    gotoxy(i,1);  write('=');
    gotoxy(i,24); write('=');
   end;
  for i := 2 to 23 do
   begin
    gotoxy(1,i);  write('‖');
    gotoxy(80,i); write('‖');
   end;
  gotoxy(1,1);   write('╔');
  gotoxy(80,1);  write('╗');
  gotoxy(1,24);  write('╚');
  gotoxy(80,24); write('╝');
  (* ========================= destiné à la phase de test ===========
  pack(cmd);
  if cmd <> ''
   then begin
         l := length(cmd);
         p := pos('@',cmd);
         if p = 0
          then begin
                nom_fic_in := cmd;
                nom_fic_out := '';
                ecran := true;
                imprimante := false;
                disque := false;
               end
          else begin
                nom_fic_in := left(cmd,p-1);
                nom_fic_out := right(cmd,l-p);
                ecran := false;
                imprimante := false;
                disque := true;
               end;
       end
   else begin
   ========================================================= *)
         ecran := true;
         imprimante := false;
         disque := false;
         nom_fic_in := '';
         nom_fic_out := '';
      (*  end; *)
```

```
window(2,1,78,24);
msg('PROLICO 1.1',2,false);
window(2,5,78,23);  (* on protege le cadre et le titre *)
gotoxy(10,1);  write('1) Nom du fichier de données : ');
gotoxy(10,3);  write('2) Désirez-vous les résultats :');
gotoxy(17,5);  write('Sur l''écran : ');
gotoxy(17,7);  write('Sur l''imprimante : ');
gotoxy(17,9);  write('Sur fichier disque : ');
gotoxy(13,10); write('(oui = nom du fichier, non = nom vide ou "//")');
gotoxy(41,1);  write(nom_fic_in);
gotoxy(31,5);
if ecran then write('o') else write('n');
gotoxy(36,7);
if imprimante then write('o') else write('n');
gotoxy(37,9);
if disque
 then write(nom_fic_out)
 else write(' ':40);
if cmd = ''
 then
  repeat
   repeat
    gotoxy(2,18); clreol;
    lis_string(41,1,nom_fic_in);
    lis_boolean(31,5,ecran);
    lis_boolean(36,7,imprimante);
    lis_string(38,9,nom_fic_out);
    gotoxy(20,18); write('Vos choix sont-ils définitifs : ');
    fini := true;
    lis_boolean(52,18,fini);
   until fini;
   pack(nom_fic_in);
   pack(nom_fic_out);
   if nom_fic_out <> '' then disque := true;
   if (not ecran) and (not imprimante) and (not disque)
    then msg('Vous devez choisir au moins une sortie des résultats',14,true)
    else begin
          gotoxy(2,14); clreol;
          gotoxy(2,15); clreol;
          gotoxy(2,16); clreol;
         end;
   if not existe(nom_fic_in)
    then msg('Le fichier ' + nom_fic_in + ' n''existe pas',11,true)
    else begin
          gotoxy(2,11); clreol;
          gotoxy(2,12); clreol;
          gotoxy(2,13); clreol;
         end;
  until (ecran or imprimante or disque) and existe(nom_fic_in);
clrscr;
msg('Traitement de "' + nom_fic_in + '"',1,false);
```

```
 if disque
  then begin
        msg('Résultats dans "' + nom_fic_out + '" ',4,false);
        assign(f_out,nom_fic_out);
        (*$i-*)
        rewrite(f_out);
        i := ioresult;
        (*$i+*)
        if i <> 0 then begin
                        disque := false;
                        erreur(73);
                       end;
       end;
 end;

(*****************************************************************************)
(* Utilisé en phase de test; affiche le contenu du fichier NOM (fichier de *)
(* données, sur les périphériques choisis                                    *)
(*****************************************************************************)

(*
 procedure affiche_contenu(nom : str80);
  var f : text;
      li : str80;
      i : byte;
  begin
   assign(f,nom);
   reset(f);
   repeat
    readln(f,li);
    ecrisln(li);
   until eof(f);
   close(f);
   ecrisln('');
   for i := 1 to 80 do ecris('-');
   ecrisln('');
   ecrisln('');
   ecrisln('');
  end;

*)

 (************************************************)
 (* fonction booléenne renvoyant true si CAR est *)
 (* l'un des séparateurs définis et faux sinon   *)
 (************************************************)

 function separateur(car : char) : boolean;
  begin
   case car of
   '(',')','[',']',',','_',';',':','=','+','-','*','\' : separateur := true;
   else separateur := false;
   end;
  end;
```

```
(*****************************************************)
(* fonction entière renvoyant le nombre de valeurs *)
(* possibles pour les éléments de la famille REF   *)
(*****************************************************)

function largeur(ref : integer) : integer;
 begin
  largeur := tab_ind[ref].borne_sup - tab_ind[ref].borne_inf + 1;
 end;

(*****************************************************************)
(* procedure lisant la ligne suivante dans le fichier de données *)
(* en passant outre des lignes vides éventuelles                 *)
(*****************************************************************)

procedure lis_ligne;
 begin
  cpt_in := 1;
  repeat
   oldstr := strin;
   readln(f_in,strin);
  until (strin <> '') or (eof(f_in));
  if strin = '' then strin := '\';  (* positionnement d'un flag *)
 end;                               (* de fin de fichier        *)

(*****************************************************************)
(* fournit dans la variable globale STRW l'ensemble de caractères*)
(* suivants en provenance du fichier de données, cet ensemble    *)
(* étant constitué d'un groupe de caractères terminés par un     *)
(* séparateur et d'un total de 80 caractères au plus             *)
(* production du message d'erreur no 67 en cas de fin de fichier *)
(* intempestive                                                  *)
(* conversion en minuscules et élimination des blancs            *)
(*****************************************************************)

procedure elem_suiv;
 begin
  strw := '';
  repeat
   cpt_in := cpt_in + 1;
   if cpt_in > length(strin)
    then lis_ligne;
   if strin[cpt_in] <> ' ' then strw := strw + lowcase(strin[cpt_in]);
  until separateur(strin[cpt_in]);
  if debug1 then writeln('!',strw,'!');
  if (strw = '\') or (strw = '') then erreur(67); (* signal eof *)
 end;
```

```
(*************************************************************************)
(* ouverture du fichier de données et lecture de la première ligne *)
(*************************************************************************)

procedure init_elem_suiv;
 var i : byte;
 begin
  (* affiche_contenu(nom_fic_in); *)
  assign(f_in,nom_fic_in);
  reset(f_in);
  oldstr := '****** debut du fichier *****';
  cpt_in := 0;
  readln(f_in,strin);
 end;

(*********************************************************)
(* procédure mettant la variable globale OK à faux si    *)
(* le dernier caractère de droite de STRG n'est pas CAR *)
(*********************************************************)

procedure ver_separateur(car : char;
                         strg : str80);
 begin
  if right(strg,1) <> car then ok := false;
 end;

(*************************************************)
(* fonction booléenne renvoyant true si le Ième   *)
(* caractère de STRG est une lettre et faux sinon *)
(*************************************************)

function anal_nom(i : integer;strg : str80) : boolean;
 begin
  anal_nom := false;
  if (strg[i] >= 'a') and (strg[i] <= 'z')
   then anal_nom := true;
 end;

(*************************************************)
(* procédure générant le message d'erreur no 1 si *)
(* la fonction précédente a renvoyé faux          *)
(* pour le traitement de STRG                     *)
(*************************************************)

procedure ver_nom(strg : str80);
 begin
  x('ver_nom');
  if anal_nom(1,strg) = false
      then erreur(1);
  y('ver nom');
 end;
```

```
(*************************************************************)
(* procédure produisant le message d'erreur no 24 en cas     *)
(* d'absence de symbole de quantification                    *)
(*************************************************************)

procedure ver_pour_tout(strg : str80);
 begin
 x('ver_pour_tout');
  if strg[1] <> '#' then erreur(24);
  y('ver_pour_tout');
 end;

(*********************************************************)
(* procédure générant le message d'erreur no 42 si STRG *)
(* amputée du dernier caractère est un mot réservé      *)
(*********************************************************)

procedure ver_mot_res(strg : str80);
 var strwrk : str80;
 begin
  x('ver_mot_res');
  strwrk := left(strg,length(strg)-1);
  if (strwrk = 'fam') or (strwrk = 'cst') or (strwrk = 'equ')
   or (strwrk = 'com') or (strwrk = 'min') or (strwrk = 'max')
   or (strwrk = 'som')
   then erreur(42);
  y('ver_mot_res');
 end;

(*************************************************************************)
(* procédure plaçant STRG1 amputée du dernier caractère dans STRG2 *)
(*************************************************************************)

procedure place_nom(strg1 : str80;
                    var strg2 : str20);
 begin
  strg2 := left(strg1,length(strg1)-1);
 end;

(*********************************************************)
(* supposant le tableau de mémorisation des familles     *)
(* correct, la procédure renvoie le message d'erreur     *)
(* no 37 si STRG amputée du dernier caractère a été      *)
(* défini comme nom de l'une des ENT premières familles  *)
(*********************************************************)

procedure ver_fam_nom(strg : str80;
                      ent : integer);
 var cpt : integer;
     strwrk : str80;
 begin
  x('ver_fam_nom');
  strwrk := left(strg,length(strg)-1);
  for cpt := 1 to ent do
   if tab_ind[cpt].nom = strwrk then erreur(37);
  y('ver_fam_nom');
 end;
```

```
{*****************************************************************}
{* procédure générant le message d'erreur no27 si STRG amputée   *}
{* du dernier caractère n'est pas un réel, rendu dans PROVISOIRE *}
{*****************************************************************}

procedure ver_type_valeur(strg : str80;
                          var provisoire : real);
 var code : integer;
     strwrk : str80;
 begin
  x('ver_type_valeur');
  strwrk := left(strg,length(strg)-1);
  if strwrk = '' then erreur(27);
  val(strwrk,provisoire,code);
  if code <> 0 then erreur(27);
  y('ver_type_valeur');
 end;

{*****************************************************************}
{* supposant le tableau de mémorisation des familles correct, *}
{* la procédure renvoie le message d'erreur no 20 si elle n'a *}
{* pu reconnaître en STRG un nom de famille, le message no 21 *}
{* si le nombre de noms de familles NB_IND dépasse le maximum *}
{* autorisé, remplit le tableau TAB et vérifie selon la valeur*}
{* de TRT qu'il n'y ait pas de répétitions indésirables de    *}
{* noms de familles, auquel cas le message no 65              *}
{*****************************************************************}

procedure ver_exist_unique(strg : str80;
                           trt : str3;
                           var nb_ind : integer;
                           var tab : tableau);
 var trouve : boolean;
     cpt : integer;
 begin
  x('ver_exist_unique');
  trouve := false;
  for cpt := 1 to nb_fam do
   if strg = tab_ind[cpt].nom
    then begin
           trouve := true;
           if nb_ind = max_indices then erreur(21);
           nb_ind := nb_ind + 1;
           tab[nb_ind] := cpt
         end;
  if not trouve then erreur(20);
  for cpt := 1 to nb_ind - 1 do
   if (trt = 'ptt') or (trt = 'som')
    then if tab[cpt] = tab[nb_ind]
           then if tab[cpt] <> 0 then erreur(65);
  y('ver_existe_unique');
 end;
```

```
(*****************************************************************)
(* supposant le tableau de mémorisation des familles correct, *)
(* la procédure renvoie le message d'erreur no 37 si elle a   *)
(* découvert que STRG amputée du dernier caractère est un nom *)
(* apparu dans une famille définie par énumération            *)
(*****************************************************************)

procedure ver_nom_par(strg : str80;
                      ent : integer);
 var cpt1,cpt2 : integer;
     strwrk : str80;
 begin
  x('ver_nom_par');
  strwrk := left(strg,length(strg)-1);
  for cpt1 := 1 to ent do
   if not tab_ind[cpt1].num
    then for cpt2 := tab_ind[cpt1].borne_inf to tab_ind[cpt1].borne_sup do
          if strwrk = mem_alpha[cpt2] then erreur(37);
  y('ver_nom_par');
 end;

(*****************************************************************)
(* supposant les tableaux TAB1 et TAB2 correctement remplis, *)
(* la procédure génère le message d'erreur no 26 si les NB1  *)
(* valeurs éventuellemnt non nulles contenues dans TAB1 ne   *)
(* sont pas un sous-ensemble des NB2 contenues dans TAB2     *)
(*****************************************************************)

procedure ver_coherence_ind(tab1,tab2 : tableau;
                            nb1,nb2 : integer);
 var cpt1,cpt2 : integer;
 begin
  x('ver_coherence_ind');
  coherence := true;
  for cpt1 := 1 to nb1 do
   if tab1[cpt1] <> 0
    then begin
          coherence := false;
          for cpt2 := 1 to nb2 do
           if tab1[cpt1] = tab2[cpt2]
            then coherence := true;
         end;
  if not coherence then erreur(26);
  y('ver_coherence_ind');
 end;
```

```
(*************************************************************)
(* supposant le tableau de mémorisation des familles correct, *)
(* la procédure génère le message d'erreur no 39 si l'entier  *)
(* ENTIER n'est pas compris dans les bornes de la famille NO  *)
(*************************************************************)

procedure appart_int(no,entier : integer);
 begin
 x('appart_ind');
  if (tab_ind[no].borne_inf > entier) or
     (tab_ind[no].borne_sup < entier)
   then erreur(39);
   y('appart_ind');
 end;

(*************************************************************************)
(* fonction entière supposant correct l'enregistrement des familles *)
(* renvoyant le numero d'élément de MEM_ALPHA où elle a découvert   *)
(* STRG comme nom apparaissant dans l'énumération de la famille de  *)
(* numéro NO ou le message d'erreur no 41 si elle ne l'a pas trouvé *)
(*************************************************************************)

function appart_enum(no : integer;strg : str20) : integer;
 var cpt : integer;
     trouve : boolean;
 begin
  x('appart_enum');
  trouve := false;
  for cpt := tab_ind[no].borne_inf to tab_ind[no].borne_sup do
   if mem_alpha[cpt] = strg
    then begin
          trouve := true;
          appart_enum := cpt;
         end;
  if not trouve then erreur(41);
  y('appart_enum');
 end;

(*************************************************************)
(* supposant le tableau des familles correct, la procédure *)
(* génère le message d'erreur no 54 si les familles de     *)
(* numéro REF1 et REF2 ne sont pas toutes deux définies    *)
(* intervalle ou n'ont pas le même nombre d'éléments       *)
(*************************************************************)

procedure verif_compat(ref1,ref2 : integer);
 begin
 x('verif_compat');
  if tab_ind[ref1].nom = tab_ind[ref2].nom then erreur(54);
  if largeur(ref1) <> largeur(ref2)
   then erreur(54);
  if (not tab_ind[ref1].num) or (not tab_ind[ref2].num)
   then erreur(54);
   y('verif_compat');
 end;
```

```
(*************************************************************)
(* procédure de remplissage des tableaux techniques de       *)
(* reconnaissance de notations mathématiques particulières   *)
(* TAB contient les références aux familles, TABCODE et      *)
(* TABVAL sont les tableaux à remplir, STRG contient la      *)
(* chaîne à analyser, NB est le numéro de l'indice, CONTR1   *)
(* permet d'identifier les éléments concernés, CONTR2        *)
(* permet de distinguer les cas de décalage                  *)
(* production des messages d'erreur no 60 en cas de valeur   *)
(* non valide, no 26 en cas d'incohérence                    *)
(*************************************************************)

procedure ext2_trav(strg : str80;
                    var tab,tabcode,tabval : tableau;
                    nb,contr1,contr2 : integer);
 var cpt,entier,code : integer;
     nom_fam : boolean;
 begin
  x('ext2_trav_deb');
  nom_fam := false;
  if strg = '' then erreur(60);
  val(strg,entier,code);
  if code = 0
   then begin
         appart_int( tab[nb],entier);
         tabcode[nb] := 1;      (* valeur partic d'une famille *)
         tabval[nb] := entier; (* définie par intervalle      *)
        end
   else if contr1 = 1  (* en provenance de cst ou var *)
         then for cpt := 1 to nb_fam do
               if strg = tab_ind[cpt].nom
                then begin
                      tabcode[nb] := 3;  (* décalage *)
                      tabval[nb] := cpt;
                      nom_fam := true;
                      case contr2 of
                       1 : verif_compat(tab[nb],tabval[nb]);
                       0 : if tab[nb] <> tabval[nb]
                            then erreur(26);
                      end;
                     end;
  if (not nom_fam) and (code <> 0)
   then begin
         tabcode[nb] := 2;                    (* valeur partic d'une famille *)
         tabval[nb] := appart_enum(tab[nb],strg);(* définie par énumération *)
        end;
  y('ext2_trav_deb');
 end;
```

```
(*************************************************************)
(* procédure de reconnaissance de notations mathématiques    *)
(* particulières pour constantes et variables                *)
(* STRG contient la chaîne courante amputée du dernier       *)
(* caractère, NB contient le numéro d'indice,                *)
(* TAB contient les références aux familles, TABCODE et      *)
(* TABVAL contenant le reflet d'éventuelles particularités   *)
(* production du message d'erreur no 53 en cas d'analyse     *)
(* impossible de STRG, no 26 en cas d'incohérence            *)
(* CODE permet de distinguer le cas des constantes du cas    *)
(* des variables                                             *)
(*************************************************************)

procedure cst_var_ext1_trav(strg : str80;
                            var tab,tabcode,tabval : tableau;
                            var nb : integer;
                            code : integer);
 var cpt,cnt : integer;
     strtrav1,strtrav2 : str80;
     separe : boolean;
 begin
  x('cst_var_ext1_trav');
  separe := false;
  for cnt := 1 to length(strg) do
   if strg[cnt] = '>'              (* division de la chaîne courante *)
    then if strg[cnt+1] = '<'
          then begin
                strtrav1 := left(strg,cnt-1);
                strtrav2 := right(strg,length(strg) - cnt - 1);
                separe := true;
               end
          else erreur(53);
  case code of
   0 : begin
        if tab[nb] = 0 then erreur(26);
        if not separe
         then begin
               if strg <> tab_ind[tab[nb]].nom then erreur(26);
               end
         else begin
                if strtrav1 <> tab_ind[tab[nb]].nom then erreur(26);
                ext2_trav(strtrav2,tab,tabcode,tabval,nb,1,1);
               end;
        end;
    1 : if not separe then ver_exist_unique(strg,'par',nb,tab)
                      else begin
                            ver_exist_unique(strtrav1,'par',nb,tab);
                            ext2_trav(strtrav2,tab,tabcode,tabval,nb,1,1);
                           end;
   end;
  y('cst_var_ext1_trav');
 end;
```

```
(*************************************************************)
(* procédure de reconnaissance des noms de familles de       *)
(* quantification, incluant le traitement des éventuelles    *)
(* notations particulières                                   *)
(* CODE indique s'il s'agit de quantification rencontrée     *)
(* lors des déclarations de constantes ou de contraintes,    *)
(* NB est le numéro de l'indice,                             *)
(* TAB contient les références aux familles, TABCODE et      *)
(* TABVAL contenant le reflet d'éventuelles particularités   *)
(* production du message d'erreur no 21 en cas de            *)
(* dépassement du nombre maximal de noms de familles permis  *)
(*************************************************************)

procedure trt_ind_ptt(var nb : integer;
                      var tab,tabcode,tabval : tableau;
                      code : integer);
 var strwrk : str80;
 begin
  x('trt_ind_ptt');
  nb := 0;
  repeat
   ver_pour_tout(strw);
   strwrk := copy(strw,2,length(strw)-2);
   case code of
    0 : ver_exist_unique(strwrk,'ptt',nb,tab);
    1 : som_ptt_ext1_trav(strwrk,tab,tabcode,tabval,nb,0);
   end;
   elem_suiv;
  until (nb > max_indices) or (right(strw,1) <> ',');
  if nb > max_indices then erreur(21);
  y('trt_ind_ptt');
 end;
```

```
(*************************************************************)
(* procédure de reconnaissance de notations mathématiques    *)
(* particulières pour sommation et quantification            *)
(* STRG contient la chaîne courante amputée du dernier       *)
(* caractère, NB contient le numéro d'indice,                *)
(* TAB contient les références aux familles, TABCODE et      *)
(* TABVAL contenant le reflet d'éventuelles particularités   *)
(* CONTROLE est un paramètre utilisé pour identifier les     *)
(* cas de décalage                                           *)
(* production du message d'erreur no 53 en cas d'analyse     *)
(* impossible de STRG                                        *)
(*************************************************************)

procedure som_ptt_ext1_trav(strg : str80;
                            var tab,tabcode,tabval : tableau;
                            var nb : integer;
                            controle : integer);
 var cpt,cnt : integer;
     strtrav1,strtrav2 : str80;
     separe : boolean;
 begin
 x('som_ptt_ext1_trav');
  separe := false;
  for cnt := 1 to length(strg) do
   if strg[cnt] = '<'           (* division de la chaîne en cas *)
    then if strg[cnt+1] = '>'   (* de notation particulière     *)
          then begin
                strtrav1 := left(strg,cnt-1);
                strtrav2 := right(strg,length(strg) - cnt - 1);
                separe := true;
               end
          else erreur(53);
  if not separe then ver_exist_unique(strg,'par',nb,tab)
                else begin
                      ver_exist_unique(strtrav1,'par',nb,tab);
                      ext2_trav(strtrav2,tab,tabcode,tabval,nb,controle,1);
                     end; (* analyse de la particularité *)
 y('som_ptt_ext1_trav');
 end;
```

```
(*************************************************************)
(* procédure de reconnaissance des noms de familles dans     *)
(* les parenthèses des différents éléments qui peuvent en    *)
(* comporter, incluant le traitement des éventuelles         *)
(* notations particulières                                   *)
(* CODE indique de quel élément il s'agit (constante,        *)
(* variable ou somme), NB est le numéro de l'indice,         *)
(* TAB contient les références aux familles, TABCODE et      *)
(* TABVAL contenant le reflet d'éventuelles particularités   *)
(* CONTROLE est un paramètre utilisé pour identifier les     *)
(* cas de décalage                                           *)
(* production des messages d'erreur no 19 en cas d'erreur    *)
(* de séparateur, no 21 en cas de dépassement du nombre      *)
(* maximal de noms de familles permis                        *)
(*************************************************************)

procedure trt_ind_par(var nb : integer;
                      var tab,tabcode,tabval : tableau;
                      code,controle : integer);
 var strwrk : str80;
 begin
 x('trt_ind_par');
  nb := 0;
  elem_suiv;
  repeat
   ver_separateur(',',strw);
   if not ok then begin
                   ok := true;
                   ver_separateur(')',strw);
                  end;
   if not ok then erreur(19);
   strwrk := copy(strw,1,length(strw)-1);
   case code of
    0 : ver_exist_unique(strwrk,'par',nb,tab);
    1 : som_ptt_ext1_trav(strwrk,tab,tabcode,tabval,nb,controle);
    2 : cst_var_ext1_trav(strwrk,tab,tabcode,tabval,nb,1);
   end;
   elem_suiv;
  until ((right(strw,1) <>',') and (right(strw,1) <>')')) or
        (nb > max_indices);
  if nb > max_indices then erreur(21);
  y('trt_ind_par');
 end;
```

```
(*************************************************************************)
(*                                                                       *)
(*                Reconnaissance de la syntaxe                           *)
(*                                                                       *)
(*************************************************************************)

overlay procedure compilation;  (* overlay de niveau 1 *)

(*****************************)
(* introduction des familles *)
(*****************************)

overlay procedure intro_index;  (* overlay de niveau 2 *)
  var i,                  (* compteur de familles *)
      j : integer;        (* compteur d'enregistrement pour énumérations *)
      moins_inf,moins_sup : boolean; (* info sur le signe des bornes  *)

  (*************************************************************)
  (* supposant l'enregistrement correctement effectué jusqu'ici, *)
  (* la procedure génère le message d'erreur no 37 si STRG      *)
  (* amputée du dernier caractère est une string déjà apparue   *)
  (* dans l'énumération de la famille encours de déclaration    *)
  (*************************************************************)

  procedure ver_act_nom_par(strg : str80);
   var cpt : integer;
       strwrk : str80;
   begin
    x('ver_act_nom_par');
    strwrk := left(strg,length(strg)-1);
     for cpt := tab_ind[i].borne_inf to j-1 do
      if strwrk = mem_alpha[cpt] then erreur(37);
     y('ver_act_nom_par');
   end;

  (*************************************************************)
  (* procédure générant le message d'erreur no 6 si la borne *)
  (* supérieure est inférieure ou égale à la borne supérieure*)
  (*************************************************************)

  procedure comp_bornes;
   begin
   x('comp_bornes');
    if tab_ind[i].borne_sup <= tab_ind[i].borne_inf
     then erreur(6);
     y('comp_bornes');
   end;
```

```
(*****************************************************************)
(* procédure générant le message d'erreur no 7 si STRG amputée *)
(* du dernier caractère n'est pas une valeur entière           *)
(* attribue le signe convenable à la borne supérieure          *)
(*****************************************************************)

procedure ver_bsup(strg : str80);
 var code : integer;
     strwrk : str80;
 begin
  x('ver_bsup');
  strwrk := left(strg,length(strg)-1);
  if strwrk = '' then erreur(7);
  val(strwrk,tab_ind[i].borne_sup,code);
  if code <> 0
   then erreur(7);
  if moins_sup then tab_ind[i].borne_sup := - tab_ind[i].borne_sup;
   y('ver_bsup');
 end;

(* idem pour la borne inférieure avec message no 8 *)

procedure ver_binf(strg : str80);
 var code : integer;
     strwrk : str80;
 begin
 x('ver_binf');
  strwrk := copy(strg,1,length(strg)-1);
  if strwrk = '' then erreur(8);
  val(strwrk,tab_ind[i].borne_inf,code);
  if code <> 0
   then erreur(8);
  if moins_inf then tab_ind[i].borne_inf := - tab_ind[i].borne_inf;
   y('ver_binf');
 end;

(*****************************************************************)
(* vérification de la syntaxe de déclaration d'une famille définie *)
(* par intervalle : production des messages d'erreurs no 9, 10, 56 *)
(* et 57 en cas d'erreur de séparateur ou de valeur introduite ou  *)
(* remplissage du record adéquat                                   *)
(*****************************************************************)

procedure index_numerique;
 begin
  x('index_numerique');
  tab_ind[i].num := true;
  elem_suiv;
  ver_separateur('_',strw);
  if not ok
   then begin
         ok := true;
         if (strw <> '-') and (strw <> '+') then erreur(56);
         if strw = '-' then moins_inf := true;
         elem_suiv;
         ver_separateur('_',strw);
         if not ok then erreur(9);
        end;
```

```
  ver_binf(strw);
  elem_suiv;
  ver_separateur(']',strw);
  if not ok
   then begin
         ok := true;
         if (strw <> '-') and (strw <> '+') then erreur(57);
         if strw = '-' then moins_sup := true;
         elem_suiv;
         ver_separateur(']',strw);
         if not ok then erreur(10);
        end;
  ver_bsup(strw);
  comp_bornes;
  y('index_numerique');
 end;

(*****************************************************************)
(* procédure de vérification et de stockage de STRG amputée   *)
(* du dernier caractère dans le tableau de mémorisation des   *)
(* noms énumérés, production du message d'erreur no 31 en cas *)
(* de dépassement de ce tableau                                *)
(*****************************************************************)

procedure stockage(strg : str80);
 begin
  x('stockage');
  if j > max_alpha then erreur(31);
  ver_nom(strg);
  ver_fam_nom(strg,i);
  ver_act_nom_par(strg);
  ver_mot_res(strg);
  mem_alpha[j] := left(strw,length(strw)-1);
  j := j + 1;
  elem_suiv;
  y('stockage');
 end;

(* idem pour le dernier STRG à traiter avec *)
(* enregistrement de la borne supérieure    *)

procedure stockfin(strg : str80);
 begin
 x('stockfin');
  if j > max_alpha then erreur(31);
  ver_nom(strg);
  ver_fam_nom(strg,i);
  ver_act_nom_par(strg);
  ver_mot_res(strg);
  mem_alpha[j] := left(strw,length(strw)-1);
  tab_ind[i].borne_sup := j;
  j := j + 1;
  y('stockfin');
 end;
```

```
(*************************************************************************)
(* vérification de la syntaxe de déclaration d'une famille définie *)
(* par énumération : production des messages d'erreurs no 12 et 22 *)
(* en cas d'erreur de séparateur ou appel à stockage                *)
(*************************************************************************)

procedure index_alphanum;
 var fini : boolean;
 begin
 x('index_alphanum');
  tab_ind[i].num := false;
  tab_ind[i].borne_inf := j;
  elem_suiv;
  repeat
   ver_separateur(',',strw);
   if not ok then erreur(22);
   stockage(strw);
  until (right(strw,1) <> ',');
  ver_separateur(')',strw);
  if not ok then erreur(12);
  stockfin(strw);
  y('index_alphanum');
 end;

(*************************************************************)
(* agencement de l'alternative intervalle/énumération,   *)
(* production des messages d'erreur no 14, 15, 16 en cas *)
(* d'erreur de séparateur et 58 en cas de dépassement    *)
(* du nombre maximal permis de noms de familles          *)
(*************************************************************)

begin (* intro_index *)
x('intro_index');
 i := 1;
 j := 1;
 elem_suiv;
 repeat
  if i > max_fam then erreur(58);
  moins_inf := false;
  moins_sup := false;
  ver_separateur('=',strw);
  if not ok then erreur(14);
  ver_nom(strw);
  ver_fam_nom(strw,i-1);
  ver_nom_par(strw,i-1);
  ver_mot_res(strw);
  place_nom(strw,tab_ind[i].nom);
  elem_suiv;
  ver_separateur('(',strw);
  if not ok
   then begin
         ok := true;
         ver_separateur('[',strw);
         if not ok then erreur(15);
         index_numerique;
        end
    else index_alphanum;
```

```
  elem_suiv;
  if strw <> ';' then erreur(16);
  i := i+1;
  elem_suiv;
 until (right(strw,1) = ':');
 nb_fam := i-1;
 nb_elem_fam := j-1;
 y('intro_index');
end;
```

```
(*****************************************)
(* introduction des valeurs de constantes  *)
(* toutes les procédures et fonctions      *)
(* supposent correct l'enregistrement des  *)
(* familles                                *)
(* production des messages no 23, 14 et 22 *)
(* en cas d'erreur de séparateur, no 26 en *)
(* cas d'incohérence et 59 si le nbre de   *)
(* noms de constantes dépasse la maximum   *)
(*****************************************)

overlay procedure intro_cst;     (* overlay de niveau 2 *)

  var i,                                          (* compteur de constantes *)
      j,cpt_cst,                                  (* compteurs de stockage  *)
      cpt,nb_ind_ptt,nb_ind_par : integer;        (* compteurs de travail   *)
      strwrk : str80;
      reecr,                                      (* info sur la re-déclaration *)
                                                  (* d'une constante            *)
      coherence : boolean;
      sauve : tableau;                            (* stockage provisoire des    *)
                                                  (* familles de quantification *)

  (*************************************************************************)
  (* enregistrement explicite des valeurs d'une constante en tenant *)
  (* des diverses combinaisons possibles pour la quantification     *)
  (* production du message d'erreur no 29 en cas d'erreur dans le   *)
  (* nombre de valeurs de constante fournies                        *)
  (*************************************************************************)

  procedure trt_scd_mbre;
   var jlocal : integer;
       reel : real;

   (*********************************************************)
   (* procédure d'enregistrement d'une valeur de constante   *)
   (* production du message d'erreur no 11 si le nombre de   *)
   (* valeurs à enregister dépasse le nombre attendu pour    *)
   (* cette constante ou du message no 32 si le nombre total *)
   (* de valeurs mémorisées dépasse le maximum permis        *)
   (*********************************************************)

   procedure stock;
    begin
    x('stock');
     mem_const[jlocal] := reel;
     jlocal := jlocal + 1;
     if jlocal > max_reel then erreur(32);
     cpt_cst := cpt_cst - 1;
     if cpt_cst < 0 then erreur(11);
     if (cpt_cst = 0) and (right(strw,1) <> ';') then erreur(11);
     y('stock');
    end;
```

```
(*************************************************************)
(* procédure d'enregistrement d'une valeur de constante      *)
(* déjà présente dans le tableau PERIODE éléments avant,     *)
(* cet enregistrement devant s'effectuer PERIODE fois        *)
(* dans des positions consécutives                           *)
(* production du message d'erreur no 11 si le nombre de      *)
(* valeurs à enregister dépasse le nombre attendu pour       *)
(* cette constante ou du message no 32 si le nombre total    *)
(* de valeurs mémorisées dépasse le maximum permis           *)
(*************************************************************)

procedure stockbis(periode : integer);
 var iloc : integer;
 begin
  x('stockbis');
  for iloc := 1 to periode do
   begin
    mem_const[jlocal] := mem_const[jlocal - periode];
    jlocal := jlocal + 1;
    if jlocal > max_reel then erreur(32);
    cpt_cst := cpt_cst - 1;
    if cpt_cst < 0 then erreur(11);
    if (cpt_cst = 0) and (right(strw,1) <> ';') then erreur(11);
   end;
  y('stockbis');
 end;

(*****************************************************************)
(* procédure commune aux différents cas possibles permettant     *)
(* d'enregistrer N fois la valeur réelle contenue dans la        *)
(* chaîne courante en des positions consécutives et cela NB      *)
(* fois; production des messages d'erreur no 16 en cas           *)
(* d'erreur de séparateur et 29 en cas de nombre de valeurs      *)
(* fournies incorrect                                            *)
(*****************************************************************)

procedure commune(n,nb : integer);
 var cnt1,cnt2 : integer;
     st2 : string[1];
 begin
   x('commune_cst1');
   cnt2 := 1;
   while cnt2 <= nb do
    begin
     elem_suiv;
     if (strw = '-') or (strw = '+')
      then begin
            st2 := strw;
            elem_suiv;
            strw := st2 + strw;
           end;
     ver_separateur(',',strw);
     if ok then begin
                 ver_type_valeur(strw,reel);
                 for cnt1 := 1 to n do stock;
                end
           else begin
                 ok := true;
                 ver_separateur(';',strw);
```

```
                if ok then begin
                                ver_type_valeur(strw,reel);
                                for cnt1 := 1 to n do stock;
                                if (cpt_cst > 0) and (cnt2 <> nb)
                                 then erreur(29);
                               end
                           else erreur(16);
                end;
     cnt2 := cnt2 + 1;
    end;
   y('commune_cst1');
 end;

(*************************************************************)
(* procédure traitant le cas des constantes ayant un seul *)
(* nom de famille dans les parenthèses                       *)
(*************************************************************)

procedure un;
 var ref1,ref11 : integer;

 (* il y a une quantification sur ce nom de famille *)

 procedure pour_i;
  var repet : integer;
  begin
  x('un_i');
   repet := largeur(ref1);
   commune(repet,1);
   y('un_i');
  end;

 (* il n'y a pas de quantification sur ce nom de famille *)

 procedure pour_rien;
  var tot : integer;
  begin
  x('un_rien');
   tot := largeur(ref11);
   commune(1,tot);
   y('un_rien');
  end;

 begin (* un *)
 x('un_cst1');
  ref1 := tab_cst[i].no_ent_ptt[1];
  ref11 := tab_cst[i].no_ent_par[1];
  if ref1 <> 0
   then pour_i
   else pour_rien;
   y('un_cst1');
 end;
```

```
 (*************************************************************)
 (* procédure traitant le cas des constantes ayant deux       *)
 (* noms de familles dans les parenthèses                     *)
 (*************************************************************)

procedure deux;
 var ref1,ref2,ref11,ref22 : integer;

 (* il y a une quantification sur le second *)

 procedure pour_j;
  var repet,tot : integer;
  begin
  x('deux_j');
   repet := largeur(ref22);
   tot := largeur(ref11);
   commune(repet,tot);
   y('deux_j');
  end;

 (* il y a une quantification sur le premier *)

 procedure pour_i;
  var repet,tot,cnt : integer;
  begin
  x('deux_i');
   repet := largeur(ref1);
   tot := largeur(ref22);
   repeat
    commune(1,tot);
    for cnt := 1 to repet-1 do stockbis(tot);
   until ((right(strw,1) = ';') or (cpt_cst < 0));
   y('deux_i');
  end;

 (* il y a une quantification sur les deux *)

 procedure pour_ij;
  var repet : integer;
  begin
  x('deux_ij');
   repet := largeur(ref2) * largeur(ref1);
   commune(repet,1);
   y('deux_ij');
  end;

 (* il n'y a pas de quantification *)

 procedure pour_rien;
  var tot : integer;
  begin
  x('deux_rien');
   tot := largeur(ref11) * largeur(ref22);
   commune(1,tot);
   y('deux_rien');
  end;
```

```
begin (* deux *)
 x('deux_cst1');
 ref1 := tab_cst[i].no_ent_ptt[1];
 ref2 := tab_cst[i].no_ent_ptt[2];
 ref11 := tab_cst[i].no_ent_par[1];
 ref22 := tab_cst[i].no_ent_par[2];
 if (ref1 <> 0) and (ref2 <> 0)
  then pour_ij
  else if (ref1 <> 0) and (ref2 = 0)
         then if (ref1 = ref11) and (ref1 = ref22)
                then pour_ij
                else if ref1 = ref11 then pour_i
                                     else pour_j
         else pour_rien;
 y('deux_cst1');
end;

(*****************************************************************)
(* procédure traitant le cas des constantes ayant trois          *)
(* noms de familles dans les parenthèses                         *)
(*****************************************************************)

procedure trois;
 var ref1,ref2,ref3,ref11,ref22,ref33 : integer;

 (* il y a une quantification sur le premier *)

 procedure pour_i;
  var repet,tot,cnt : integer;
  begin
  x('3_i');
   repet := largeur(ref1);
   tot := largeur(ref22) * largeur(ref33);
   repeat
    commune(1,tot);
    for cnt := 1 to repet-1 do stockbis(tot);
   until ((right(strw,1) = ';') or (cpt_cst < 0));
   y('3_i');
  end;

 (* il y a une quantification sur le second *)

 procedure pour_j;
  var repet,tot,cnt : integer;
  begin
  x('3_j');
   repet := largeur(ref22);
   tot := largeur(ref33);
   repeat
    commune(1,tot);
    for cnt := 1 to repet-1 do stockbis(tot);
   until ((right(strw,1) = ';') or (cpt_cst < 0));
   y('3_j');
  end;
```

```
(* il y a une quantification sur le troisième *)

procedure pour_k;
 var repet,tot : integer;
 begin
 x('3_k');
  tot := largeur(ref11) * largeur(ref22);
  repet := largeur(ref33);
  commune(repet,tot);
  y('3_k');
 end;

(* il y a une quantification sur le premier et le second *)

procedure pour_ij;
 var repet,tot,cnt : integer;
 begin
 x('3_ij');
  repet := largeur(ref2) * largeur(ref1);
  tot := largeur(ref33);
  repeat
   commune(1,tot);
   for cnt := 1 to repet-1 do stockbis(tot);
  until ((right(strw,1) = ';') or (cpt_cst <  0));
  y('3_ij');
 end;

(* il y a une quantification sur le second et le troisième *)

procedure pour_jk;
 var repet,tot : integer;
 begin
 x('3_jk');
  repet := largeur(ref22) * largeur(ref33);
  tot := largeur(ref11);
  commune(repet,tot);
  y('3_jk');
 end;

(* il y a une quantification sur le premier et le troisième *)

procedure pour_ik;
 var repet,tot1,tot2,tot3,cnt : integer;
 begin
 x('3_ik');
  tot3 := largeur(ref22);
  tot2 := largeur(ref11);
  tot1 := largeur(ref22) * largeur(ref33);
  repet := largeur(ref33);
  repeat
   for cnt := 1 to tot3 do commune(repet,1);
   for cnt := 1 to tot2-1 do stockbis(tot1);
  until ((right(strw,1) = ';') or (cpt_cst < 0));
  y('3_ik');
 end;
```

```
(* il y a une quantification sur les trois *)

procedure pour_ijk;
 var repet : integer;
 begin
 x('3_ijk');
  repet := largeur(ref1) * largeur(ref2) * largeur(ref3);
  commune(repet,1);
  y('3_ijk');
 end;

(* il n'y a pas de quantification *)

procedure pour_rien;
 var tot : integer;
 begin
 x('3_rien');
  tot := largeur(ref11) * largeur(ref22) * largeur(ref33);
  commune(1,tot);
  y('3_rien');
 end;

(*************************************************************)
(* procédures d'orientation du traitement en fonction du *)
(* contenu des tableaux générés par la reconnaissance    *)
(*************************************************************)

procedure trois1;
 begin
  x('31');
  if (ref1 = ref11) and (ref1 = ref22) and (ref1 = ref33)
   then pour_ijk
   else if ((ref1 = ref11) and (ref1 = ref22))
         then pour_ij
         else if (ref1 = ref11) and (ref1 = ref33)
               then pour_ik
               else if (ref1 = ref22) and (ref1 = ref33)
                     then pour_jk
                     else if (ref1 = ref11) and (ref1 <> ref22)
                               and (ref1 <> ref33)
                           then pour_i
                           else if (ref1 <> ref11) and (ref1 = ref22)
                                     and (ref1 <> ref33)
                                 then pour_j
                                 else pour_k;
  y('31');
 end;

procedure trois2;
 begin
 x('32');
  if ((ref1 = ref11) and (ref2 = ref22))
     or ((ref1 = ref22) and (ref2 = ref11))
   then if (ref1 <> ref33) and (ref2 <> ref33)
         then pour_ij
         else pour_ijk
```

```
   else if ((ref1 = ref11) and (ref2 = ref33))
           or ((ref1 = ref33) and (ref2 = ref11))
         then if (ref1 <> ref22) and (ref2 <> ref22)
                then pour_ik
                else pour_ijk
         else if ((ref1 = ref22) and (ref2 = ref33))
                 or ((ref1 = ref33) and (ref2 = ref22))
                then if (ref1 <> ref11) and (ref2 <> ref11)
                       then pour_jk
                       else pour_ijk;
 y('32');
 end;

begin (* trois *)
 x('trois_cst');
 ref1 := tab_cst[i].no_ent_ptt[1];
 ref2 := tab_cst[i].no_ent_ptt[2];
 ref3 := tab_cst[i].no_ent_ptt[3];
 ref11 := tab_cst[i].no_ent_par[1];
 ref22 := tab_cst[i].no_ent_par[2];
 ref33 := tab_cst[i].no_ent_par[3];
 if (ref1 <> 0) and (ref2 = 0) and (ref3 = 0)
  then trois1
  else if (ref1 <> 0) and (ref2 <> 0) and (ref3 = 0)
         then trois2
         else if (ref1 <> 0) and (ref2 <> 0) and (ref3 <> 0)
                then pour_ijk
                else pour_rien;
 y('trois_cst');
end;

 begin (* trt_scd_mbre *)
  x('trt_scd-mbre');
  jlocal := j;
  tab_cst[i].debut := jlocal;
  case nb_ind_par of
   0 : commune(1,1);
   1 : un;
   2 : deux;
   3 : trois;
  end;
  if cpt_cst > 0 then erreur(29);
  tab_cst[i].total := jlocal - j;
  j := jlocal;
  elem_suiv;
  y('trt_scd_mbre');
 end;
```

```
(*****************************************************************)
(* calcul du nombre total de valeurs de constantes à mémoriser *)
(* en fonction des familles apparaissant dans les parenthèses  *)
(*****************************************************************)

procedure calcul_place_totale;
 var cpt,ref : integer;
 begin
  x('calc_place_tot');
  for cpt := 1 to nb_ind_par do
   begin
    ref := tab_cst[i].no_ent_par[cpt];
    cpt_cst := cpt_cst * largeur(ref);
   end;
  y('calc_place_tot');
 end;

(*****************************************************************)
(* procédure produisant le message d'erreur no 26 en cas       *)
(* d'incohérence entre les familles apparaissant dans les      *)
(* parenthèses de la constante et celles apparaissant dans la  *)
(* quantification                                               *)
(*****************************************************************)

procedure verif_fam;
 var cpt,cnt : integer;
     trouve : boolean;
 begin
  for cpt := 1 to max_indices do
   begin
    trouve := true;
    if sauve[cpt] <> 0 (* famille apparaissant dans la quantification *)
     then begin
           trouve := false;
           cnt := 1;
           repeat
            if mem[cnt] = sauve[cpt]    (* famille apparaissant dans les *)
                                        (* parenthèses et quantification *)
             then if code_mem[cnt] = 3 (* famille présente par son nom  *)
                   then trouve := true;
            cnt := cnt + 1;
           until trouve or (cnt > max_indices);
          end;
    if not trouve then erreur(26);
   end;
 end;

(*****************************************************************)
(* procédure gérant la réecriture sur la constante la plus     *)
(* récemment déclarée en fonction du type de réecriture exigé, *)
(* différentes combinaisons étant offertes sur les noms de     *)
(* familles, leurs valeurs particulières et la quantification  *)
(*****************************************************************)

procedure reecriture;
 var partiel,modif : integer;
     valeur : real;
```

```
(************************************************)
(* enregistrement d'une valeur déjà enregistrée *)
(* PERIODE éléments plus tôt                    *)
(************************************************)

procedure enr_bis(periode : integer);
 var depart : integer;
 begin
  x('enr_bis');
  depart := modif;
  mem_const[depart] := mem_const[depart - periode];
  depart := depart + 1;
  y('enr_bis');
 end;

(*************************************************************)
(* enregistrement de NB2 valeurs de constantes, chacune NB1  *)
(* fois à DISTANCE éléments d'intervalle                     *)
(* production des messages d'erreur no 11 et 29 en cas       *)
(* d'erreur au niveau du nombre de valeurs fournies et no    *)
(* 16 en cas d'erreur de séparateur                          *)
(*************************************************************)

procedure commune(nb1,nb2,distance : integer);
 var cnt1,cnt2,depart : integer;
     st2 : char;

 (****************************************)
 (* enregistrement d'une valeur NB1 fois *)
 (* à DISTANCE éléments d'intervalle     *)
 (****************************************)

 procedure enregistre;
  begin
   x('enregistre');
   for cnt1 := 1 to nb1 do
    begin
     mem_const[depart] := valeur;
     depart := depart + distance;
    end;
   y('enregistre');
  end;

 begin (* commune *)
 x('commune_cst2');
  depart := modif;
  cnt2 := 1;
  while cnt2 <= nb2 do
   begin
    elem_suiv;
    if (strw = '-') or (strw = '+')
     then begin
           st2 := strw;
           elem_suiv;
           strw := st2 + strw;
          end;
    ver_separateur(',',strw);
```

```
    if ok then begin
                ver_type_valeur(strw,valeur);
                enregistre;
                partiel := partiel - 1;
                if partiel <= 0 then erreur(11);
               end
          else begin
                ok := true;
                ver_separateur(';',strw);
                if ok then begin
                            ver_type_valeur(strw,valeur);
                            enregistre;
                            partiel := partiel - 1;
                            if partiel > 0 then erreur(29);
                           end
                      else erreur(16);
               end;
    cnt2 := cnt2 + 1;
   end;
  y('commune_cst2');
 end;

(*************************************************************)
(* procedure traitant le cas de réecriture d'une constante *)
(* ayant un seul nom de famille dans les parenthèses       *)
(*************************************************************)

procedure un;
 var ref,no1 : integer;
 begin
  x('1-reecr');
  ref := tab_cst[i-1].no_ent_par[1];
  no1 := largeur(ref);
  modif := tab_cst[i-1].debut - tab_ind[ref].borne_inf + mem[1];
  if code_mem[1] <> 3
   then commune(1,1,0)
   else if sauve[1] <> 0 then commune(no1,1,1)
                         else commune(1,no1,1);
  y('1-reecr');
 end;

(*************************************************************)
(* procedure traitant le cas de réecriture d'une constante *)
(* ayant un deux noms de familles dans les parenthèses     *)
(*************************************************************)

procedure deux;
 var ref1,ref2,no1,no2,d1,d2,cpt : integer;
 begin
 x('2-reecr');
  ref1 := tab_cst[i-1].no_ent_par[1];
  ref2 := tab_cst[i-1].no_ent_par[2];
  d1 := tab_ind[ref1].borne_inf;
  d2 := tab_ind[ref2].borne_inf;
  no1 := largeur(ref1);
  no2 := largeur(ref2);
```

```
 if (code_mem[1] <> 3) and (code_mem[2] <> 3) (* deux valeurs  *)
  then begin                                  (* particulières *)
        modif := tab_cst[i-1].debut + mem[2] - d2 + (mem[1] - d1) * no2;
        commune(1,1,0)
       end
  else if code_mem[1] <> 3 (* une valeur comme premier indice   *)
        then begin         (* et un nom de famille comme second *)
              modif := tab_cst[i-1].debut + no2 * (mem[1] - d1);
              if sauve[1] <> 0 (* ce nom de famille apparaît dans # *)
               then commune(no2,1,1)
               else commune(1,no2,1);
             end
         else if code_mem[2] <> 3 (* inversément *)
              then begin
                    modif := tab_cst[i-1].debut + mem[2] - d2;
                    if sauve[1] <> 0
                     then commune(no1,1,no2)
                     else commune(1,no1,no2);
                    end
              else begin (* deux noms de familles *)
                    modif := tab_cst[i-1].debut;
                    if sauve[1] = ref1        (* traités selon leur    *)
                     then if sauve[2] = ref2 (* apparition dans les # *)
                           then commune(no1 * no2,1,1)
                           else for cpt := 1 to no2 do
                                 begin
                                  commune(no1,1,no2);
                                  modif := modif + 1;
                                 end
                     else if sauve[1] = ref2
                           then if sauve[2] = ref1
                                 then commune(no1 * no2 ,1,1)
                           else for cpt := 1 to no1 do
                                 begin
                                  commune(no2,1,1);
                                  modif := modif + no2;
                                 end
                           else commune(1,no1 * no2,1);
                   end;
  y('2-reecr');
 end;

(*************************************************************)
(* procedure traitant le cas de réecriture d'une constante *)
(* ayant un trois noms de familles dans les parenthèses    *)
(*************************************************************)

procedure trois;
 var ref1,ref2,ref3,no1,no2,no3,d1,d2,d3 : integer;
```

```
(*********************************************************)
(* chacune des procédures suivantes correspond à un cas *)
(* particulier de configuration de réecriture           *)
(*********************************************************)

procedure trois1;
 begin
  x('31-reecr');
  modif := tab_cst[i-1].debut + mem[3] - d3 + (mem[2] - d2) * no3
           + (mem[1] - d1) * no2 * no3;
  commune(1,1,0);
  y('31-reecr');
 end;

procedure trois2;
 begin
  x('32-reecr');
  modif := tab_cst[i-1].debut + (mem[1] - d1) * no2 * no3
           + (mem[2] - d2) * no3;
  if sauve[1] <> 0 (* famille dans les # *)
   then commune(no3,1,1)
   else commune(1,no3,1);
  y('32-reecr');
 end;

procedure trois3;
 begin
  x('33-reecr');
  modif := tab_cst[i-1].debut + (mem[1] - d1) * no2 * no3 + mem[3] - d3;
  if sauve[1] <> 0 (* famille dans les # *)
   then commune(no2,1,no3)
   else commune(1,no2,no3);
  y('33-reecr');
 end;

procedure trois4;
 begin
  x('34-reecr');
  modif := tab_cst[i-1].debut + (mem[2] - d2) * no3 + mem[3] - d3;
  if sauve[1] <> 0 (* famille dans les # *)
   then commune(no1,1,no2 * no3)
   else commune(1,no1,no2 * no3);
  y('34-reecr');
 end;

procedure trois5;
 var cpt : integer;
 begin
  x('35-reecr');
  modif := tab_cst[i-1].debut + (mem[1] - d1) * no2 * no3;
  if sauve[1] = ref2              (* familles traitées selon leur *)
   then if sauve[2] = ref3        (* apparition dans les #        *)
         then commune(no2 * no3,1,1)
         else for cpt := 1 to no3 do
               begin
                commune(no2,1,no3);
                modif := modif + 1;
               end
```

```
   else if sauve[1] = ref3
         then if sauve[2] = ref2
                then commune(no2 * no3,1,1)
                else for cpt := 1 to no2 do
                       begin
                        commune(no3,1,1);
                        modif := modif + no3;
                       end
           else commune(1,no2 * no3,1);
   y('35-reecr');
  end;

 procedure trois6;
  var cpt,cnt : integer;
  begin
   x('36-reecr');
   modif := tab_cst[i-1].debut + (mem[2] - d2) * no3;
   if sauve[1] = ref1                (* familles traitées selon leur *)
    then if sauve[2] = ref3          (* apparition dans les #        *)
          then begin
                commune(no3,1,1);
                modif := modif + no2 * no3;
                for cpt := 1 to no1-1 do
                 begin
                  for cnt := 1 to no3 do
                   begin
                    enr_bis(no2 * no3);
                    modif := modif + 1;
                   end;
                  modif := modif + (no2 - 1) * no3;
                 end;
               end
          else for cpt := 1 to no3 do
                begin
                 commune(no1,1,no2 * no3);
                 modif := modif + 1;
                end
     else if sauve[1] = ref3
           then if sauve[2] = ref1
                 then begin
                       commune(no3,1,1);
                       modif := modif + no2 * no3;
                       for cpt := 1 to no1-1 do
                        begin
                         for cpt := 1 to no3 do
                          begin
                           enr_bis(no2 * no3);
                           modif := modif + 1;
                          end;
                         modif := modif + (no2 - 1) * no3;
                        end;
                      end
                 else for cpt := 1 to no1 do
                       begin
                        commune(no3,1,1);
                        modif := modif + no2 * no3;
                       end
```

```
        else for cpt := 1 to no1 do
               begin
                commune(1,no3,1);
                modif := modif + no2 * no3;
               end;
  y('36-reecr');
 end;

procedure trois7;
 var cpt : integer;
 begin
  x('37-reecr');
  modif := tab_cst[i-1].debut + mem[3] - d3;
  if sauve[1] = ref1          (* familles traitées selon leur *)
   then if sauve[2] = ref2    (* apparition dans les #        *)
          then commune(no2 * no1,1,no3)
          else for cpt := 1 to no2 do
                 begin
                  commune(no1,1,no2 * no3);
                  modif := modif + no3;
                 end
   else if sauve[1] = ref2
          then if sauve[2] = ref1
                 then commune(no2 * no1,1,no3)
                 else for cpt := 1 to no1 do
                        begin
                         commune(no2,1,no3);
                         modif := modif + no2 * no3;
                        end
          else commune(1,no1 * no2,no3);
  y('37-reecr');
 end;

 (* il y a quantification sur le premier et le second indices *)

 procedure pour_ij;
  var cpt : integer;
  begin
   for cpt := 1 to no3 do
    begin
     commune(no1 * no2,1,no3);
     modif := modif + 1;
    end;
  end;

 (* il y a quantification sur le premier et le troisième indices *)

 procedure pour_ik;
  var cpt,cnt : integer;
  begin
   for cpt := 1 to no2 do
     begin
      commune(no3,1,1);
      modif := modif + no3;
     end;
```

```
   for cpt := 1 to no1-1 do
    for cnt := 1 to no2 * no3 do
     begin
      enr_bis(no2 * no3);
      modif := modif + 1;
     end;
  end;

 (* il y a quantification sur le premier indice *)

 procedure pour_i;
  var cpt : integer;
  begin
   for cpt := 1 to no2 * no3 do
    begin
     commune(no1,1,no2 * no3);
     modif := modif + 1;
    end;
  end;

 (* il y a quantification sur le second et le troisième indices *)

 procedure pour_jk;
  var cpt : integer;
  begin
   for cpt := 1 to no1 do
    begin
     commune(no2 * no3,1,1);
     modif := modif + no2 * no3;
    end;
  end;

 (* il y a quantification sur le second indice *)

 procedure pour_j;
  var cpt,cnt : integer;
  begin
   for cpt := 1 to no1 do
    begin
     for cnt := 1 to no3 do
      begin
       commune(no2,1,no3);
       modif := modif + 1;
      end;
     modif := modif + no3;
    end;
  end;

 (* il y a quantification sur le troisième indice *)

 procedure pour_k;
  var cpt : integer;
  begin
   for cpt := 1 to no1 * no2 do
    begin
     commune(no3,1,1);
     modif := modif + no3;
    end;
  end;
```

```
procedure trois8;
 begin
  x('38-reecr');
  modif := tab_cst[i-1].debut;              (* familles traitées selon leur *)
  if sauve[1] <> 0                          (* apparition dans les #        *)
   then if sauve[2] <> 0
         then if sauve[3] <> 0
               then commune(no1 * no2 * no3,1,1)
               else if sauve[1] = ref1
                     then if sauve[2] = ref2
                           then pour_ij
                           else if sauve[2] = ref3
                                 then pour_ik
                                 else pour_i
                     else if sauve[1] = ref2
                           then if sauve[2] = ref1
                                 then pour_ij
                                 else if sauve[2] = ref3
                                       then pour_jk
                                       else pour_j
                           else if sauve[1] = ref3
                                 then if sauve[2] = ref1
                                       then pour_ik
                                       else if sauve[2] = ref2
                                             then pour_jk
                                             else pour_k
                                 else commune(1,no1 * no2 * no3,1)
         else if sauve[1] = ref1
               then pour_i
               else if sauve[1] = ref2
                     then pour_j
                     else if sauve[1] = ref3
                           then pour_k
                           else commune(1,no1 * no2 * no3,1)
   else commune(1,no1 * no2 * no3,1);
  y('38-reecr');
 end;

 begin (* trois *)
  x('trois-reecr');
  ref1 := tab_cst[i-1].no_ent_par[1];
  ref2 := tab_cst[i-1].no_ent_par[2];
  ref3 := tab_cst[i-1].no_ent_par[3];
  d1 := tab_ind[ref1].borne_inf;
  d2 := tab_ind[ref2].borne_inf;
  d3 := tab_ind[ref3].borne_inf;
  no1 := largeur(ref1);
  no2 := largeur(ref2);
  no3 := largeur(ref3);
  if (code_mem[1] <> 3) and (code_mem[2] <> 3) and (code_mem[3] <> 3)
   then trois1 (* trois valeurs particulières *)
   else if (code_mem[1] <> 3) and (code_mem[2] <> 3)
         then trois2 (* deux premiers indices sont des val partic *)
         else if (code_mem[1] <> 3) and (code_mem[3] <> 3)
               then trois3 (* premier et troisième sont des val partic *)
               else if (code_mem[2] <> 3) and (code_mem[3] <> 3)
                     then trois4 (* deuxième et troisième sont val partic *
                     else if code_mem[1] <> 3 (* premier est val partic *)
```

```
                              then trois5
                              else if code_mem[2] <> 3 (* deuxième *)
                                    then trois6
                                    else if code_mem[3] <> 3 then trois7
   (* troisième *)                                              else trois8;
   (* aucun     *)
   y('trois-reecr');
  end;

(*******************************************)
(* réecriture d'une constante sans indices *)
(*******************************************)

procedure zero;
 begin
  x('0-reecr');
  modif := tab_cst[i-1].debut;
  commune(1,1,0);
  y('0-reecr');
 end;

 (*****************************************************************)
 (* calcule le nombre de valeurs de constantes à enregistrer  *)
 (* pour la réecriture d'une constante, en tenant compte des  *)
 (* possibilités de quantification                            *)
 (*****************************************************************)

 procedure calcul_place_partielle;
  var cpt,cnt : integer;
      trouve : boolean;
  begin
   x('calc_place_part');
   partiel := 1;
   for cpt := 1 to max_indices do
    if code_mem[cpt] = 3 (* nom de famille *)
     then begin
           cnt := 1;
           trouve := false;
           repeat
            if sauve[cnt] = mem[cpt]
             then trouve := true; (* cette famille apparaît dans les # *)
            cnt := cnt + 1;
           until (trouve) or (cnt > max_indices);
           if not trouve then partiel := partiel * largeur(mem[cpt]);
          end;
   y('calc_place_part');
  end;

 begin (* reecriture *)
  x('reecriture');
  calcul_place_partielle;
  case nb_ind_par of
   0 : zero;
   1 : un;
   2 : deux;
   3 : trois;
  end;
  elem_suiv;
  y('reecriture');
 end;
```

```
(*****************************************************************)
(* traitement des familles apparaissant dans les déclarations  *)
(* de constantes comportant au moins un nom de famille         *)
(* production du message d'erreur no 19 en cas d'erreur de     *)
(* séparateur, traitement de la chaîne courante en tant que    *)
(* nom de famille                                              *)
(*****************************************************************)

procedure contenu_par;
 var strwrk : str20;
 begin
  x('contenu_par');
  nb_ind_par := 0;
  elem_suiv;
  repeat
   ver_separateur(',',strw);
   if not ok then begin
                   ok := true;
                   ver_separateur(')',strw);
                   if not ok then erreur(19);
                  end;
   nb_ind_par := nb_ind_par + 1;
   strwrk := copy(strw,1,length(strw)-1);
   ext2_trav(strwrk,tab_cst[i-1].no_ent_par,code_mem,mem,nb_ind_par,1,0);
   elem_suiv;
  until (nb_ind_par > max_indices) or ((right(strw,1) <> ',') and
        (right(strw,1) <> ')'));
  if nb_ind_par > max_indices then erreur(21);
  if strw <> '=' then erreur(14);
  verif_fam;
  reecriture;
  y('contenu-par');
 end;

(*****************************************************************)
(* procédure permettant de vérifier si STRG amputée du dernier *)
(* caractère constitue une réecriture sur une même constante,  *)
(* une double déclaration ou une nouvelle déclaration          *)
(* production du message no 37 en cas de double déclaration,   *)
(* du message no 23 en cas d'erreur de séparateur, orientation *)
(* de la suite du traitement                                   *)
(*****************************************************************)

procedure ver_dble_decl(strg : str80);
 var cpt : integer;
     strwrk : str80;
 begin
 x('ver_dble_decl_cst');
  for cpt := 1 to max_indices do mem[cpt] := 0;
  strwrk := copy(strg,1,length(strg)-1);
  for cpt := 1 to i-2 do
   if strwrk = tab_cst[cpt].nom
    then erreur(37);
  if strwrk = tab_cst[i-1].nom
   then begin
         reecr := true;
         ver_separateur('(',strg);
```

```
          if not ok
           then begin
                 ok := true;
                 ver_separateur('=',strw);
                 if not ok then erreur(23);
                 nb_ind_par := 0;
                 verif_fam;
                 reecriture;
                end
           else contenu_par;
          end;
  y('ver_dble_decl-cst');
 end;

begin (* intro_cst *)
x('intro-cst');
 for i := 1 to max_cst do              (* initialisations *)
  for cpt := 1 to max_indices do
   begin
    tab_cst[i].no_ent_ptt[cpt] := 0;
    tab_cst[i].no_ent_par[cpt] := 0;
   end;
  for cpt := 1 to max_indices do
   begin
    val_fictif[cpt] := 0;
    code_fictif[cpt] := 0;
   end;
 i := 1;
 j := 1;
 elem_suiv;
 repeat
  if i > max_cst then erreur(59);
  nb_ind_ptt := 0;
  nb_ind_par := 0;
  reecr := false;
  cpt_cst := 1;
  for cpt := 1 to max_indices do sauve[cpt] := 0;
  if strw[1] = '#' then
   begin
    ver_separateur(',',strw);
    if not ok then erreur(22);
    trt_ind_ptt(nb_ind_ptt,sauve,code_fictif,val_fictif,0);
                (* stockage provisoire des familles de quantification *)
   end;
  if i <> 1 then ver_dble_decl(strw);
  if not reecr (* nouvelle constante *)
   then begin
         for cpt := 1 to max_indices do
          tab_cst[i].no_ent_ptt[cpt] := sauve[cpt];
          ver_nom(strw);
          ver_mot_res(strw);
          ver_fam_nom(strw,nb_fam);
          ver_nom_par(strw,nb_fam);
          place_nom(strw,tab_cst[i].nom);
          ver_separateur('=',strw);
```

```
        if ok then begin (* constante sans noms de familles *)
                    ver_coherence_ind(tab_cst[i].no_ent_ptt,
                                      tab_cst[i].no_ent_par,
                                      nb_ind_ptt,nb_ind_par);
                    calcul_place_totale;
                    trt_scd_mbre; (* enregistrement de la valeur *)
                   end
              else begin (* constante avec noms de familles *)
                    ok := true;
                    ver_separateur('(',strw);
                    if not ok then erreur(23);
                    trt_ind_par(nb_ind_par,tab_cst[i].no_ent_par,
                                code_fictif,val_fictif,0,0);
                    (* enregistrement des noms de familles *)
                    if nb_ind_par < nb_ind_ptt then erreur(26);
                    ver_separateur('=',strw);
                    if not ok then erreur(14);
                    ver_coherence_ind(tab_cst[i].no_ent_ptt,
                                      tab_cst[i].no_ent_par,
                                      nb_ind_ptt,nb_ind_par);
                    calcul_place_totale;
                    trt_scd_mbre; (* enregistrement des valeurs *)
                   end;
         i := i + 1;
        end;
 until (right(strw,1) = ':');
 nb_cst := i-1;
 nb_elem_cst := j-1;
 y('intro_cst');
end;
```

```
(*************************************************************)
(* procédure prenant en charge la reconnaissance de la       *)
(* fonction objectif et des contraintes et l'enregistrement  *)
(* de toutes les informations qui y sont relatives           *)
(* toutes les procédures et fonctions supposent corrects     *)
(* l'enregistrement des familles et des constantes           *)
(*************************************************************)

overlay procedure intro_equ;      (* overlay de niveau 2 *)
 var refcst,refvar,               (* références à la constante et  *)
                                  (* à la variable rencontrée dans *)
                                  (* l'élément de somme courant    *)
     nb_ind_som,          (* compteur de nombre de familles dans  *)
     nb_ind_var,          (* la somme, la variable et la quanti-  *)
     nb_ind_ptt,          (* fication (contrainte uniquement) de  *)
                          (* l'élément de somme courant           *)
     ivar,i : integer;    (* compteur du nombre de variables et   *)
                          (* du nombre de contraintes             *)
     sortie,trouve,                (* booléens de travail contenant*)
     signal_plus,signal_moins,     (* des informations relatives   *)
     signal_fois : boolean;        (* à l'état de l'élément de     *)
                                   (* somme courant                *)
     fictif_var2,fictif_code2_var,fictif_val2_var : tableau;

 (*********************************************)
 (* procédure enlevant de STRWRK son dernier *)
 (* caractère si ce caractère est '>' ou '<' *)
 (*********************************************)

 procedure precaution(var strwrk : str80);
  begin
   if (right(strwrk,1) = '>') or (right(strwrk,1) = '<')
    then strwrk := left(strwrk,length(strwrk)-1);
  end;

 (*************************************************)
 (* procédure vérifiant que CAR est un séparateur *)
 (*************************************************)

 procedure repet(car : char);
  begin
  x('repet');
   ok := true;
   ver_separateur(car,strw);
   y('repet');
  end;

 (*************************************************************)
 (* procédure de reconnaissance du séparateur entre deux      *)
 (* membres d'une contrainte et mise_à_jour du champ          *)
 (* adéquat du record, s'il y a lieu, renvoi direct vers      *)
 (* le traitement du second membre s'il y a lieu ou           *)
 (* génération du message d'erreur no 49 si la contrainte     *)
 (* n'a pas de second membre                                  *)
 (*************************************************************)
```

```
procedure sep_sec;
 var code : integer;
     car : char;
 begin
 x('sep_sec');
  if tab_equ[i].code_equ = 0
   then begin
         sortie := false;
         code := 0;
         if right(strw,1) = '='
          then begin
                code := 1;
                if length(strw) > 1
                 then begin
                       car := strw[length(strw)-1];
                       case car of
                        '>' : code := 2;
                        '<' : code := 3;
                       end;
                       if (strw = '=') or (strw = '<=') or (strw = '>=')
                        then sortie := true; (* renvoi vers SCD_MBRE *)
                      end;
               end;
         if code = 0 then erreur(49);
         tab_equ[i].code_equ := code;
        end;
  y('sep_sec');
 end;

(*************************************************************)
(* procédure générant le message d'erreur no 37 si STRG      *)
(* amputée du ou des deux derniers caractères a déjà été     *)
(* utilisé comme nom de variable parmi celles apparues       *)
(* jusqu'à présent                                           *)
(*************************************************************)

procedure ver1_var_nom(strg : str80);
 var strwrk : str80;
     cpt : integer;
 begin
  x('ver1_var_nom');
  strwrk := left(strg,length(strg)-1);
  precaution(strwrk);
  for cpt := 1 to ivar-1 do
   if tab_var[cpt].nom = strwrk then erreur(37);
  y('ver1_var_nom');
 end;

(*****************************************************************)
(* procédure permettant de conserver dans COPIE les références   *)
(* aux familles apparaissant dans TAB2 et pas dans TAB1          *)
(*****************************************************************)
```

```
procedure ver_bis_coherence_ind(tab1,tab2 : tableau;
                                var copie : tableau);
 var cpt,cnt : integer;
 begin
  x('ver_bis_coherence_ind');
  for cpt := 1 to max_indices do copie[cpt] := 0;
  for cpt := 1 to max_indices do copie[cpt] := tab2[cpt];
  for cpt := 1 to max_indices do
   if copie[cpt] <> 0
    then for cnt := 1 to max_indices do
          if tab1[cnt] = copie[cpt]
           then copie[cpt] := 0;
  y('ver_bis_coherence_ind');
 end;

(*************************************************************)
(* procédure générant le message d'erreur no 37 si STRG      *)
(* amputée du ou des deux derniers caractères a déjà été     *)
(* utilisé comme nom de contrainte parmi celles apparues     *)
(* jusqu'à présent                                           *)
(*************************************************************)

procedure ver_equ_nom(strg : str80);
 var strwrk : str80;
     cpt : integer;
 begin
  x('ver_equ_nom');
  strwrk := left(strg,length(strg)-1);
  precaution(strwrk);
  for cpt := 1 to i-1 do
   if tab_equ[cpt].nom = strwrk then erreur(37);
  y('ver_equ_nom');
 end;

(*********************************************************)
(* procédure enregistrant la référence à l'éventuelle    *)
(* constante dont STRG amputée du ou des deux derniers   *)
(* caractères pourrait être le nom                       *)
(*********************************************************)

procedure ver_cst_nom(strg : str80);
 var cpt : integer;
     strwrk : str80;
 begin
  x('ver_cst_nom');
  trouve := false;
  strwrk := left(strg,length(strg)-1);
  precaution(strwrk);
  for cpt := 1 to nb_cst do
   if tab_cst[cpt].nom = strwrk
    then begin
          trouve := true;
          refcst := cpt;
         end;
  y('ver_cst_nom');
 end;
```

```
(*********************************************************************)
(* procedure de reconnaissance des noms de familles apparaissant     *)
(* dans l'élément de somme NO_ELEM au sein d'une constante ou        *)
(* d'une variable déjà apparue (selon la valeur de CODE) et          *)
(* correctement enregistrée dans TAB, CONTROLE permettant de         *)
(* distinguer fonction objectif, premier ou second membre de         *)
(* la Ième contrainte                                                *)
(*********************************************************************)

procedure rec_cst_var_ind(tab : tableau;
                          no_elem,controle,code : integer);
 var ref1,ref2,ref3,     (* références aux noms de familles *)
     nb : integer;       (* compteur de nombre de noms de familles *)
     str1,str2,str3 : str20;

 (*********************************************************************)
 (* procédure de reconnaissance proprement dite générant le message   *)
 (* d'erreur no NO en cas d'erreur sur le séparateur CAR ou appelant  *)
 (* la procédure permettant de vérifier que STRG amputée du dernier   *)
 (* caractère est un nom de famille apparaissant à bon escient        *)
 (*********************************************************************)

 procedure rec_ind(no : integer;
                   car : char;
                   strg : str20);
  var strwrk : str80;
  begin
   x('rec_ind');
   elem_suiv;
   ver_separateur(car,strw);
   if not ok then erreur(no);
   strwrk := left(strw,length(strw)-1);
   case controle of
    0 : case code of
         1 : cst_var_ext1_trav(strwrk,tab_cst[refcst].no_ent_par,
                               fo.code_cst[no_elem],fo.val_cst[no_elem],nb,0);
         2 : cst_var_ext1_trav(strwrk,tab_var[refvar].no_ent_par,
                               fo.code_var[no_elem],fo.val_var[no_elem],nb,0);
        end;
    1 : case code of
         1 : cst_var_ext1_trav(strwrk,tab_cst[refcst].no_ent_par,
                               tab_equ[i].code1_cst[no_elem],
                               tab_equ[i].val1_cst[no_elem],nb,0);
         2 : cst_var_ext1_trav(strwrk,tab_var[refvar].no_ent_par,
                               tab_equ[i].code1_var[no_elem],
                               tab_equ[i].val1_var[no_elem],nb,0);
        end;
    2 : case code of
         1 : cst_var_ext1_trav(strwrk,tab_cst[refcst].no_ent_par,
                               tab_equ[i].code2_cst[no_elem],
                               tab_equ[i].val2_cst[no_elem],nb,0);
         2 : cst_var_ext1_trav(strwrk,tab_var[refvar].no_ent_par,
                               fictif_code2_var,fictif_val2_var,nb,0);
        end;
   end;
   nb := nb + 1;
  y('rec_ind');
  end;
```

```
begin (* rec_cst_var_ind *)
 x('rec_cst_var_ind');
 nb := 1;
 ref1 := tab[1];
 ref2 := tab[2];
 ref3 := tab[3];
 if (ref1 <> 0) and (ref2 <> 0) and (ref3 <> 0)
  then begin
         str1 := tab_ind[ref1].nom;
         rec_ind(22,',',str1);
         str2 := tab_ind[ref2].nom;
         rec_ind(22,',',str2);
         str3 := tab_ind[ref3].nom;
         rec_ind(19,')',str3);
        end
  else if (ref1 <> 0) and (ref2 <> 0)
         then begin
                str1 := tab_ind[ref1].nom;
                rec_ind(22,',',str1);
                str2 := tab_ind[ref2].nom;
                rec_ind(19,')',str2);
               end
         else if ref1 <> 0
                then begin
                       str1 := tab_ind[ref1].nom;
                       rec_ind(19,')',str1);
                      end
                else begin
                       ver_separateur('(',strw);
                       if ok then erreur(26);
                      end;
 if (ref1 <> 0) or (ref2 <> 0) or (ref3 <> 0)
  then elem_suiv;
 if right(strw,1) = '='                    (* état de l'élément de *)
  then sep_sec                             (* somme courant        *)
  else if right(strw,1) = '+'
         then signal_plus := true
         else if right(strw,1) = '-'
                then signal_moins := true
                else if right(strw,1) = '*' then signal_fois := true;
 y('rec_cst_var_ind');
end;

(*****************************************************************)
(* enregistrement des informations relatives à STRG amputée du *)
(* dernier ou des deux derniers caractères si celle-ci          *)
(* représente un nom de variable                                *)
(* apparaissant pour la première fois alors que l'on se trouve *)
(* dans le traitement de l'élément de somme NO_ELEM de la       *)
(* fonction objectif, du premier ou du second membre de la Ième*)
(* contrainte, selon la valeur de CODE                          *)
(* s'il ne s'agit pas d'une variable apparaissant pour la       *)
(* première fois, on s'oriente vers la reconnaissance des noms *)
(* de familles y apparaissant éventuellement                    *)
(*****************************************************************)
```

```
procedure ver2_var_nom(strg : str80;
                       no_elem,code : integer);
 var strwrk : str80;
     cpt : integer;
     stop : boolean;

 (*****************************************************************)
 (* procédure d'enregistrement des informations relatives à une *)
 (* variable apparaissant pour la première fois ou production   *)
 (* du message d'erreur no 68 en cas de dépassement du nombre   *)
 (* maximal de noms de variables autorisé                       *)
 (*****************************************************************)

 procedure interne;
  begin
   tab_var[ivar].nom := strwrk;
   refvar := ivar;
   ivar := ivar + 1;
   if ivar > max_var then erreur(68);
   nb_ind_var := 0;
   ver_separateur('(',strw);
   if ok
    then case code of
          0 : trt_ind_par(nb_ind_var,tab_var[refvar].no_ent_par,
                          fo.code_var[no_elem],fo.val_var[no_elem],2,0);
          1 : trt_ind_par(nb_ind_var,tab_var[refvar].no_ent_par,
                          tab_equ[i].code1_var[no_elem],
                          tab_equ[i].val1_var[no_elem],2,1);
          2 : trt_ind_par(nb_ind_var,tab_var[refvar].no_ent_par,
                          fictif_code2_var,fictif_val2_var,2,1);

         end
    else ok := true;
  end;

 begin (* ver2_var_nom *)
  x('ver2_var_nom');
  strwrk := left(strg,length(strg)-1);
  precaution(strwrk);
  if ivar = 1
   then interne
   else begin
         cpt := 1;
         stop := false;
         repeat
          if tab_var[cpt].nom = strwrk
           then begin
                 refvar := cpt;
                 rec_cst_var_ind(tab_var[refvar].no_ent_par,no_elem,code,2);
                 stop := true;
                end;
          cpt := cpt + 1;
          until (stop) or (cpt > ivar - 1);
          if not stop then interne;
         end;
  y('ver2_var_nom');
 end;
```

```
(*****************************************************************)
(* procédure de vérification de la cohérence de l'élément de     *)
(* somme NO_ELEM apparu dans la fonction objectif, le premier    *)
(* ou le second membre de la Ièmè contrainte, selon la valeur    *)
(* de CONTROLE                                                   *)
(* orientation en fonction des éléments effectivement présents   *)
(* dans l'élément de somme (constante, variable, somme)          *)
(*****************************************************************)

procedure verif_coherence(no_elem,controle : integer);
 var reste_ind,reste_bis_ind : tableau;

 (*****************************************************************)
 (* procédure générant le message d'erreur no 26 et no 65 en cas  *)
 (* d'incohérence de la déclaration d'un élément de somme         *)
 (* un nom de famille ne peut pas apparaître dans RESTE sauf      *)
 (* s'il correspond à une particularité >< non de décalage        *)
 (* reprise dans TABCODE                                          *)
 (* Un nom de famille ne peut pas apparaître dans TABSOM s'il     *)
 (* correspond à une particularité ><  non de décalage, sauf s'il*)
 (* est repris par ailleurs dans TAB                              *)
 (*****************************************************************)

 procedure forte_coher(tabsom,reste,tab,tabcode : tableau);
  var cpt,cnt,tel : integer;
      rien,autre : boolean;
  begin
   x('forte_coher');
   rien := true;
   autre := true;
   for cpt := 1 to max_indices do
    begin
     if reste[cpt] <> 0
      then if (tabcode[cpt] = 0) or (tabcode[cpt] = 3) then rien := false;
     if not rien then erreur(26);
    end;
   for cpt := 1 to max_indices do
    begin
     if (tabcode[cpt] = 1) or (tabcode[cpt] = 2)
      then for cnt := 1 to max_indices do
            begin
             if tabsom[cnt] = tab[cpt]
              then begin
                    autre := false;
                    for tel := 1 to max_indices do
                     if tel <> cpt
                      then if (tabsom[cnt] = tab[tel])
                                and (tabcode[tel] <> 1)
                                and (tabcode[tel] <> 2)
                                then autre := true;
                   end;
             if not autre then erreur(65);
            end;
    end;
   y('forte_coher');
  end;
```

```
(*****************************************************************)
(* procédure générant le message d'erreur no 65 et no 66 en cas *)
(* d'incohérence de la déclaration d'un élément de somme        *)
(* un nom de famille ne peut pas apparaître dans RESTE sauf     *)
(* s'il correspond à une particularité >< non de décalage       *)
(* reprise dans TABCODE1                                        *)
(* un nom de famille ne peut pas apparaître dans TABSOM s'il    *)
(* correspond à une particularité >< non de décaclage reprise à *)
(* la fois dans TABCODE1 et TABCODE2 sauf s'il apparaît par     *)
(* ailleurs dans TAB1 ou TAB2                                   *)
(*****************************************************************)

procedure autre_coher(tabsom,reste,tabcode1,tabcode2,tab1,tab2 : tableau);
 var rien,deux_egal,repete,trouve : boolean;
     cpt,cnt,tel : integer;
 begin
  x('autre coher');
  rien := true;
  for cpt := 1 to max_indices do
   begin
    if reste[cpt] <> 0
     then if (tabcode1[cpt] = 0) or (tabcode1[cpt] = 3)
            then rien := false;
    if not rien then erreur(65);
   end;
  deux_egal := true;
  for cpt := 1 to max_indices do
   begin
    if (tabcode1[cpt] = 1) or (tabcode1[cpt] = 2)
     then begin
            deux_egal := false;
            for cnt := 1 to max_indices do
             if (tabcode2[cnt] = 1) or (tabcode2[cnt] = 2)
              then if tab1[cpt] = tab2[cnt]
                     then deux_egal := true;
            if deux_egal
             then begin
                    repete := false;
                    for cnt := 1 to max_indices do
                     if cnt <> cpt
                      then if (tab1[cpt] = tab1[cnt])
                                and (tabcode1[cnt] <> 1)
                                and (tabcode1[cnt] <> 2)
                             then repete := true;
                    if not repete
                     then begin
                            for cnt := 1 to max_indices do
                             if (tab2[cnt] = tab1[cpt])
                                 and (tabcode2[cnt] <> 1)
                                 and (tabcode2[cnt] <> 2)
                              then repete := true;
                          end;
                    if not repete
                     then for tel := 1 to max_indices do
                            if tabsom[tel] = tab1[cpt]
                             then erreur(65);
                  end
```

```
            else begin
                  trouve := false;
                  for tel := 1 to max_indices do
                   if tabsom[tel] = tab1[cpt]
                    then trouve := true;
                  if not trouve then erreur(66);
                 end;
          end;
   end;
  y('autre coher');
 end;

(*************************************************************)
(* procédure générant le message d'erreur no 65 en cas       *)
(* d'incohérence de la déclaration d'un élément de somme     *)
(*************************************************************)

procedure coher_globale(tabsom,tab,tabcode : tableau;
                        code : integer);
 var cpt : integer;
     non_nul : boolean;
 begin
  x('coher_globale');
  ver_bis_coherence_ind(tabsom,tab,reste_ind);
  if controle <> 0 (* contrainte *)
   then begin
         non_nul := false;
         for cpt := 1 to max_indices do
          if reste_ind[cpt] <> 0
           then non_nul := true;
         if non_nul
          then ver_bis_coherence_ind(tab_equ[i].no_ent_ptt,reste_ind,
                                     reste_bis_ind)
          else for cpt := 1 to max_indices do reste_bis_ind[cpt] := 0;
         if code = 1 then forte_coher(tabsom,reste_bis_ind,tab,tabcode);
        end
   else if code = 1 then forte_coher(tabsom,reste_ind,tab,tabcode);
  if code = 1
   then begin
         ver_bis_coherence_ind(tab,tabsom,reste_ind);
         for cpt := 1 to max_indices do
          if reste_ind[cpt] <> 0
           then erreur(65);
        end;
  y('coher_globale');
 end;

 (*************************************************************)
 (* procédure de vérification de cohérence lorsque ni         *)
 (* constante ni variable n'est présente, production du       *)
 (* message d'erreur no 65 s'il existe un symbole de          *)
 (* sommation suivi de noms de familles                       *)
 (*************************************************************)
```

```
procedure rien_somme;
 var cpt : integer;
 begin
  x('rien somme');
  for cpt := 1 to max_indices do
   case controle of
    0 : if no_elem <> 0
         then if fo.no_ent_som[no_elem,cpt] <> 0 then erreur(65);
    1 : if no_elem <> 0
         then if tab_equ[i].no1_ent_som[no_elem,cpt] <> 0
               then erreur(65);
    2 : if no_elem <> 0
         then if tab_equ[i].no2_ent_som[no_elem,cpt] <> 0
               then erreur(65);
   end;
  y('rien somme');
 end;

(*********************************************************)
(* procédure de vérification de cohérence lorsque une    *)
(* constante seule est présente                          *)
(*********************************************************)

procedure cst_somme(code : integer);
 begin
  x('cst somme');
  case controle of
   0 : if no_elem <> 0
        then coher_globale(fo.no_ent_som[no_elem],
                           tab_cst[refcst].no_ent_par,
                           fo.code_cst[no_elem],code);
   1 : if no_elem <> 0
        then coher_globale(tab_equ[i].no1_ent_som[no_elem],
                           tab_cst[refcst].no_ent_par,
                           tab_equ[i].code1_cst[no_elem],code);
   2 : if no_elem <> 0
        then coher_globale(tab_equ[i].no2_ent_som[no_elem],
                           tab_cst[refcst].no_ent_par,
                           tab_equ[i].code1_cst[no_elem],code);
   end;
  y('cst somme');
 end;

(*********************************************************)
(* procédure de vérification de cohérence lorsque une    *)
(* variable seule est présente                           *)
(*********************************************************)

procedure var_somme(code : integer);
 begin
  x('var somme');
  case controle of
   0 : if no_elem <> 0
        then coher_globale(fo.no_ent_som[no_elem],
                           tab_var[refvar].no_ent_par,
                           fo.code_var[no_elem],code);
```

```
    1 : if no_elem <> 0
        then coher_globale(tab_equ[i].no1_ent_som[no_elem],
                           tab_var[refvar].no_ent_par,
                           tab_equ[i].code1_var[no_elem],code);
    end;
   y('var somme');
  end;

(*********************************************************)
(* procédure de vérification de cohérence d'un élément *)
(* de somme produisant le message d'erreur no 65 si un *)
(* nom de famille apparaît dans la quantification sans *)
(* apparaître dans les parenthèses de la constante ni  *)
(* dans celles de la variable                          *)
(*********************************************************)

procedure som_somme;
 var cpt,cnt : integer;
     tabsom : tableau;
     trouve : boolean;
 begin
  for cpt := 1 to max_indices do
   begin
    trouve := false;
    case controle of
     0 : if no_elem <> 0
         then tabsom[cpt] := fo.no_ent_som[no_elem,cpt];
     1 : if no_elem <> 0
         then tabsom[cpt] := tab_equ[i].no1_ent_som[no_elem,cpt];
     2 : if no_elem <> 0
         then tabsom[cpt] := tab_equ[i].no2_ent_som[no_elem,cpt];
    end;
    if tabsom[cpt] <> 0
     then begin
           for cnt := 1 to max_indices do
            begin
             if tab_cst[refcst].no_ent_par[cnt] = tabsom[cpt]
              then trouve := true;
            end;
            if not trouve
             then begin
                   for cnt := 1 to max_indices do
                    if tab_var[refvar].no_ent_par[cnt] = tabsom[cpt]
                     then trouve := true;
                  end;
            if not trouve then erreur(65);
          end;
   end;
 end;

(*********************************************************)
(* procédure de vérification de cohérence d'un élément *)
(* de somme d'une contrainte générant le message       *)
(* d'erreur no 64 si un nom de famille est présent à   *)
(* fois dans la sommation et la quantification         *)
(*********************************************************)
```

```
procedure ptt_somme;
 var cpt,cnt : integer;
 begin
  x('ptt somme');
  for cpt := 1 to max_indices do
   for cnt := 1 to max_indices do
    case controle of
     1 : if no_elem <> 0
         then if tab_equ[i].no_ent_ptt[cpt] <> 0
              then if tab_equ[i].no_ent_ptt[cpt] =
                      tab_equ[i].no1_ent_som[no_elem,cnt]
                   then erreur(64);
     2 : begin
         if no_elem <> 0
          then if tab_equ[i].no_ent_ptt[cpt] <> 0
               then if tab_equ[i].no_ent_ptt[cpt] =
                       tab_equ[i].no2_ent_som[no_elem,cnt]
                    then erreur(64);
         end;
    end;
  y('ptt somme');
 end;

(*************************************************************)
(* procédure de vérification de cohérence supplémentaire *)
(* entre noms de familles de quantification et noms de   *)
(* familles dans les parenthèses de la constante et de la*)
(* variable lorsque celles-ci sont toutes deux présentes *)
(*************************************************************)

procedure surplus;
 begin
  x('surplus');
  case controle of
   0 : if no_elem <> 0
       then begin
            autre_coher(fo.no_ent_som[no_elem],reste_ind,
                        fo.code_cst[no_elem],fo.code_var[no_elem],
                        tab_cst[refcst].no_ent_par,
                        tab_var[refvar].no_ent_par);
            autre_coher(fo.no_ent_som[no_elem],reste_ind,
                        fo.code_var[no_elem],fo.code_cst[no_elem],
                        tab_var[refvar].no_ent_par,
                        tab_cst[refcst].no_ent_par);
           end;
    1 : if no_elem <> 0
        then begin
             autre_coher(tab_equ[i].no1_ent_som[no_elem],reste_bis_ind,
                         tab_equ[i].code1_cst[no_elem],
                         tab_equ[i].code1_var[no_elem],
                         tab_cst[refcst].no_ent_par,
                         tab_var[refvar].no_ent_par);
             autre_coher(tab_equ[i].no1_ent_som[no_elem],reste_bis_ind,
                         tab_equ[i].code1_var[no_elem],
                         tab_equ[i].code1_cst[no_elem],
                         tab_var[refvar].no_ent_par,
                         tab_cst[refcst].no_ent_par);
             end;
```

```
      2 : if no_elem <> 0
           then begin
                 autre_coher(tab_equ[i].no2_ent_som[no_elem],reste_bis_ind,
                             tab_equ[i].code2_cst[no_elem],
                             fictif_code2_var,
                             tab_cst[refcst].no_ent_par,
                             fictif_var2);
                 end;
    end;
    y('surplus');
   end;

  begin (* verif_coherence *)
   x('verif_coherence');
   if controle <> 0
    then ptt_somme;
   if (refcst <> 0) and (refvar <> 0)   (* une constante et une *)
    then begin                          (* variable présentes   *)
          cst_somme(2);
          var_somme(2);
          som_somme;
          surplus;
         end
    else if (refcst <> 0) and (refvar = 0) (* une constante seule présente *)
          then cst_somme(1)
          else if (refvar <> 0) and (refcst = 0) (* une variable seule *)
                then var_somme(1)
                else if (refcst = 0) and (refvar = 0) (* ni constante, *)
                      then rien_somme;                (* ni variable   *)
   y('verif_coherence');
  end;

(*************************************************************************)
(* procédure d'"attaque" de l'élément de somme COURANT apparu dans la  *)
(* fonction objectif, le premier ou le second membre d'une contrainte  *)
(* (CONTROLE = 0,1 ou 2), dont le facteur de multiplication est FACT   *)
(* NB est le nombre de noms de familles présents dans les parenthèses  *)
(* de la sommation éventuelle                                          *)
(* vérification de cohérence de l'élément de somme précédent, s'il y a *)
(* lieu                                                                *)
(*************************************************************************)

procedure init_elem_som(var fact : table2;
                        var nb,courant : integer;
                        controle : integer);

(*************************************************************************)
(* procédure d'organisation de reconnaissance de l'élément de somme    *)
(* NO_ELEM selon que celui-ci commence par une sommation ou par un réel*)
(*************************************************************************)

 procedure elem_som(var no_elem : integer);
  var strwrk : str80;
      valeur : real;
```

```
(*********************************************************)
(* procédure mettant AUX_BOOL à faux si la chaîne de     *)
(* caractères courante n'est pas un réel, à vrai sinon   *)
(* et générant le message d'erreur no 51 si l'éventuel   *)
(* réel conduit à un élément de somme nul qui ne soit    *)
(* pas le dernier du second membre d'une contrainte      *)
(*********************************************************)

procedure proc_reel(var aux_bool : boolean);
 var code : integer;
 begin
  x('proc_reel');
  aux_bool := true;
  if strwrk = ''
   then aux_bool := false
   else begin
         val(strwrk,valeur,code);
         if code <> 0 then aux_bool := false
                      else if (valeur = 0) and (right(strw,1) <> ';')
                            then erreur(51);
        end;
  y('proc_reel');
 end;

(*********************************************************************)
(* procédure d'agencement des procédures de reconnaissance           *)
(* d'une constante et de son enregistrement                          *)
(* AUX_BOOL est faux si la string courante analysée ne fournit       *)
(* pas un nom de constante, CODE permet l'organisation de la         *)
(* reconnaissance des noms de familles en distinguant fonction       *)
(* objectif, premier et second membre d'une contrainte               *)
(*********************************************************************)

procedure proc_cste(var aux_bool : boolean;
                    code : integer);
 begin
  x('proc_cste');
  aux_bool := true;
  ver_cst_nom(strw);
  if not trouve then aux_bool := false
                else rec_cst_var_ind(tab_cst[refcst].no_ent_par,
                                     courant,code,1);
  y('proc_cste');
 end;

(*********************************************************************)
(* procédure d'agencement des procédures de reconnaissance           *)
(* d'une variable et de son enregistrement                           *)
(* production du message d'erreur no 37 si le nom choisi a           *)
(* déjà été utilisé comme nom de constante                           *)
(*********************************************************************)
```

```
procedure proc_var;
 begin
 x('proc_var');
  ver_nom(strw);
  ver_mot_res(strw);
  ver_fam_nom(strw,nb_fam);
  ver_nom_par(strw,nb_fam);
  ver_cst_nom(strw);
  if trouve then erreur(37);
  ver_equ_nom(strw);
  ver2_var_nom(strw,courant,controle);
  y('proc_var');
 end;

(*****************************************************************)
(* procédure d'orientation vers la reconnaissance d'un réel,     *)
(* d'une constante ou d'une variable selon la valeur de NB et    *)
(* obtention d'une valeur BOOL tenant compte du résultat de      *)
(* l'analyse                                                     *)
(*****************************************************************)

procedure choix(nb : integer;
                var bool : boolean);
 begin
  x('choix');
  case nb of
   1 : proc_reel(bool);
   2 : proc_cste(bool,controle);
   3 : proc_var;
  end;
  y('choix');
 end;

(*****************************************************************)
(* procédure auxiliaire pour la reconnaissance d'un réel,        *)
(* d'une constante ou d'une variable, selon la valeur de NB      *)
(* adaptant la valeur du booléen BOOL au résultat de l'analyse   *)
(* prenant en compte les séparateurs exceptionnels susceptibles  *)
(* de se présenter selon que l'on traite la fontion objectif,    *)
(* premier ou le second membre d'une contrainte                  *)
(* production des messages d'erreur no 10, 14 et 16 en cas       *)
(* d'erreur de séparateur                                        *)
(*****************************************************************)

procedure exception(nb : integer;
                    var bool : boolean);
 begin
  x('exception');
  case controle of
   0 : begin
        repet('1');
```

```
      if ok then begin
                  choix(nb,bool);
                  if bool then
                   begin
                    elem_suiv;
                    if strw <> ';' then erreur(16);
                    elem_suiv;
                   end;
                 end
            else erreur(10);
     end;
  1 : begin
       repet('=');
       if ok then begin
                   sep_sec;
                   if not sortie
                    then begin
                           if (right(strw,2) = '>=')
                             or (right(strw,2) = '<=')
                            then strwrk := left(strw,length(strw)-2)
                            else strwrk := left(strw,length(strw)-1);
                           choix(nb,bool);
                          (*  if bool then elem_suiv; *)
                           end;
                  end
             else erreur(49);
     end;
  2 : begin
       repet(';');
       if ok then begin
                   choix(nb,bool);
                   if bool then elem_suiv;
                  end
             else erreur(16);
     end;
 end;
 y('exception');
end;

(*****************************************************************)
(* procédure auxiliaire pour la reconnaissance d'un réel,        *)
(* d'une constante ou d'une variable, selon la valeur de NB      *)
(* adaptant la valeur du booléen BOOL au résultat de l'analyse *)
(*****************************************************************)

procedure commune(nb : integer;
                  var bool : boolean);

 begin
  x('commune_equ');
  repet('+');        (* séparateurs pouvant apparaître      *)
  if not ok          (* pour un réel, une cste ou une var  *)
   then begin
         repet('-');
         if not ok
          then exception(nb,bool)
          else begin
                choix(nb,bool);
                signal_moins := true;    (* mise à jour de l'état *)
               end;                      (* de l'élément de somme *)
        end
```

```
   else begin
         choix(nb,bool);
         signal_plus := true;   (* mise à jour de l'état *)
        end;                    (* de l'élément de somme *)
  y('commune_equ');
 end;

(*****************************************************)
(* fonction booléenne renvoyant la valeur vrai si    *)
(* la string courante amputée du dernier caractère   *)
(* ou des deux derniers est un réel                  *)
(*****************************************************)

function reel : boolean;
 var bool_reel : boolean;
 begin
  x('f-reel');
  reel := true;
  ver_separateur('*',strw);
  strwrk := left(strw,length(strw)-1);
  precaution(strwrk);
  if not ok
   then commune(1,bool_reel)
   else begin
         proc_reel(bool_reel);
         signal_fois := true;    (* mise à jour de l'état de *)
        end;                     (* l'élément de somme       *)
  if not sortie then reel := bool_reel;
  y('f-reel');
 end;

(*****************************************************)
(* fonction booléenne renvoyant la valeur vrai si    *)
(* la string courante amputée du dernier caractère   *)
(* ou des deux derniers est une constante            *)
(*****************************************************)

function constante : boolean;
 var bool_cste : boolean;
 begin
  x('f-cste');
  constante := true;
  ver_separateur('*',strw);     (* constante sans parenthèses *)
  strwrk := left(strw,length(strw)-1);
  precaution(strwrk);
  if not ok
   then begin
         repet('(');            (* constante avec parenthèses *)
         if not ok
          then commune(2,bool_cste)
          else proc_cste(bool_cste,controle);
        end
   else begin
         proc_cste(bool_cste,controle);
         signal_fois := true;     (* mise à jour de l'état de *)
        end;                      (* l'élément de somme       *)
  constante := bool_cste;
  y('f-cste');
 end;
```

```
(*****************************************************)
(* fonction booléenne renvoyant toujours la valeur *)
(* vrai à moins d'une sortie en erreur dans les    *)
(* procédures appelées                               *)
(*****************************************************)

function variable : boolean;
 var bool_var : boolean;
 begin
  x('f-var');
  ver_separateur('(',strw);
  strwrk := left(strw,length(strw)-1);
  precaution(strwrk);
  if not ok
   then commune(3,bool_var)
   else proc_var;
  variable := true;
  y('f-var');
 end;

(*****************************************************)
(* procédure de traitement et d'enregistrement des *)
(* noms de familles apparaissant dans la sommation *)
(*****************************************************)

procedure somme;
 begin
 x('somme');
  case controle of
   0 : trt_ind_par(nb,fo.no_ent_som[no_elem],fo.code_som[no_elem],
                   fo.val_som[no_elem],1,0);
   1 : trt_ind_par(nb,tab_equ[i].no1_ent_som[no_elem],
                   tab_equ[i].code1_som[no_elem],
                   tab_equ[i].val1_som[no_elem],1,1);
   2 : trt_ind_par(nb,tab_equ[i].no2_ent_som[no_elem],
                   tab_equ[i].code2_som[no_elem],
                   tab_equ[i].val2_som[no_elem],1,1);
  end;
  y('somme');
 end;

(*************************************************************)
(* procédure d'enregistrement de la référence à la constante*)
(* d'un élément de somme à l'endroit convenable              *)
(*************************************************************)

procedure enr_cst;
 begin
  x('enr_cst');
  case controle of
   0 : fo.cst[no_elem] := refcst;
   1 : tab_equ[i].cst1[no_elem] := refcst;
   2 : tab_equ[i].cst2[no_elem] := refcst;
  end;
  y('enr_cst');
 end;
```

```
(*****************************************************************)
(* procédure d'enregistrement de la référence à la variable *)
(* d'un élément de somme à l'endroit convenable                  *)
(*****************************************************************)

procedure enr_var;
 begin
  x('enr_var');
  case controle of
   0 : fo.variable[no_elem] := refvar;
   1 : tab_equ[i].var1[no_elem] := refvar;
  end;
  y('enr_var');
 end;

(*****************************************************************)
(* procédure d'agencement du passage d'un élément de somme  *)
(* au suivant (si on a un signe "+" ou "-") ou du passage   *)
(* entre éléments au sein d'un même élément de somme        *)
(* (si on a un signe "*") incluant la vérification de       *)
(* cohérence de l'élément de somme complètement traité et   *)
(* mise à 1 ou -1 du multiplicateur de l'élément de somme   *)
(* suivant, s'il y a lieu (signe "+" ou "-")                *)
(* production du message d'erreur no 46 en cas de dépasse-  *)
(* ment du nombre maximal d'éléments de somme permis        *)
(*****************************************************************)

procedure mise_en_place;
 begin
  x('mise_en_place');
  if (length(strw) = 1) and not signal_plus
      and not signal_moins and not signal_fois
   then elem_suiv;
  if signal_plus   (* signe "+" *)
   then begin
         signal_plus := false;
         courant := courant + 1;
         if courant > max_som then erreur(46);
         fact[courant] := 1;
         if length(strw) > 1
          then verif_coherence(courant-1,controle);
         refvar := 0;  (* vérification de cohérence et remise à 0 *)
         refcst := 0;
         elem_suiv;
        end
   else if signal_moins  (* signe "-" *)
         then begin
               signal_moins := false;
               courant := courant + 1;
               if courant > max_som then erreur(46);
               fact[courant] := -1;
               if length(strw) > 1
                then verif_coherence(courant-1,controle);
               refvar := 0;  (* vérification de cohérence et remise à 0 *)
               refcst := 0;
               elem_suiv;
              end
```

```
        else if signal_fois (* signe "*" *)
              then begin
                    signal_fois := false;
                    elem_suiv;
                   end;
  y('mise_en_place');
 end;

(*****************************************************************)
(* procédure d'agencement de l'ordre dans lequel les éléments *)
(* d'un même élément de somme doivent apparaître et être      *)
(* reconnus                                                   *)
(* production du message d'erreur no 50 si une variable       *)
(* apparaît dans le second membre d'une contrainte, no 63 si  *)
(* un réel apparaît seul dans le premier membre et no 62 si   *)
(* une constante apparaît seule dans le premier membre        *)
(* enregistrement des valeurs réelles s'il y a lieu           *)
(* cas où l'élément de somme commence par une sommation       *)
(*****************************************************************)

procedure petit_morceau;
 begin
  x('petit morceau');
  if strwrk <> 'som'
   then if not constante
         then begin
               if controle = 2
                then begin
                      if variable then erreur(50);
                     end
                else begin
                      if variable then enr_var;
                     end;
              end
         else begin
               if controle = 1
                then begin
                      if not signal_fois then erreur(62);
                     end;
                enr_cst;
               end
   else begin
         somme;
         ver_separateur('(',strw);
         if not ok
          then begin
                ok := true;
                if not reel
                 then begin
                       if not sortie
                        then begin
                              if not constante
                               then begin
                                     if controle = 2
                                      then begin
                                            if variable then erreur(50);
                                           end
                                      else begin
```

```
                                        if variable then enr_var;
                                       end;
                                end
                              else begin
                                    if controle = 1
                                     then begin
                                            if not signal_fois
                                             then erreur(62);
                                           end;
                                     enr_cst;
                                    end;
                           end
                         else begin
                           end;
                      end
                  else begin
                        if controle = 1
                         then begin
                              if not signal_fois then erreur(63);
                              fact[no_elem] := fact[no_elem] * valeur;
                              end
                         else begin
                               fact[no_elem] := fact[no_elem] * valeur;
                              end;
                         end;
             end
        else begin
              if not constante
               then begin
                     if controle = 2
                      then begin
                            if variable then erreur(50);
                           end
                      else begin
                            if variable then enr_var;
                           end;
                    end
               else begin
                     if controle = 1
                      then begin
                            if not signal_fois then erreur(62);
                           end;
                     enr_cst;
                    end;
              end;
       end;
 y('petit morceau');
end;
```

```
(*****************************************************************)
(* procédure d'agencement de l'ordre dans lequel les éléments *)
(* d'un même élément de somme doivent apparaître et être      *)
(* reconnus                                                   *)
(* production du message d'erreur no 50 si une variable       *)
(* apparaît dans le second membre d'une contrainte, no 63 si  *)
(* un réel apparaît seul dans le premier membre, no 62 si     *)
(* une constante apparaît seule dans le premier membre et no  *)
(* 48 en cas d'erreur de séparateur                           *)
(* enregistrement des valeurs réelles s'il y a lieu           *)
(* cas où l'élément de somme commence par un réel             *)
(*****************************************************************)

procedure gros_morceau;
 begin
  x('gros morceau');
  if not reel
   then if not sortie
          then begin
                 if strwrk <> 'som'
                  then begin
                         if not constante
                          then begin
                                 if controle = 2  (* 2ème membre *)
                                  then begin
                                         if variable then erreur(50);
                                        end
                                  else begin
                                         if variable then enr_var;;
                                        end;
                                 end
                          else begin
                                 if controle = 1  (* 1er membre *)
                                  then begin
                                         if not signal_fois then erreur(62);
                                        end;
                                 enr_cst;
                                end;
                       end
                  else begin
                         ver_separateur('(',strw);
                         if not ok then erreur(48);
                         somme;
                         if not reel
                          then begin
                                 if not sortie
                                  then begin
                                         if not constante
                                          then begin
                                                 if controle = 2
                                                  then begin
                                                         if variable
                                                          then erreur(50);
                                                        end
                                                  else begin
                                                         if variable then enr_var
                                                        end
                                                end
                                          else begin
```

```
                                                    if controle = 1
                                                     then begin
                                                           if not signal_fois
                                                            then erreur(62);
                                                           end;
                                                    enr_cst;
                                                   end
                                          end
                                     else begin
                                          end
                                end
                           else begin
                                 if not sortie
                                  then begin
                                        if controle = 1
                                         then begin
                                               if not signal_fois
                                                then erreur(63);
                                               end;
                                        fact[no_elem] := fact[no_elem] * valeur
                                        end;
                                end;
                      end;
               end
          else begin
               end
    else if not sortie then begin
                            if controle = 1
                             then begin
                                   if not signal_fois
                                    then erreur(63);
                                  end;
                            fact[no_elem] := fact[no_elem] * valeur;
                           end;
   y('gros morceau');
  end;

begin (* elem_som *)
  x('elem_som');
  mise_en_place;
  ok := true;
  if (right(strw,1) <> ',') and (right(strw,1) <> ':')
   then begin
         strwrk := left(strw,length(strw)-1);
         precaution(strwrk);
         ver_separateur('(',strw);
         if not ok then begin
                         ok := true;
                         gros_morceau;
                        end
                   else begin
                         petit_morceau;
                        end;
        end;
   y('elem_som');
end;
```

```
begin (* init_elem_som *)
 x('init_elem_som');
 if strw <> '+'
  then if right(strw,1) <> '+'
       then if strw <> '-'
            then if right(strw,1) <> '-'
                 then begin
                       if courant = 0 then courant := courant + 1;
                       if fact[courant] = 0 then fact[courant] := 1;
                       elem_som(courant);
                      end
                 else begin
                       if courant = 0 then courant := courant + 1;
                       if fact[courant] = 0 then fact[courant] := 1;
                       if courant < max_som
                        then fact[courant + 1] := -1;
                       elem_som(courant);
                      end
            else begin
                  signal_moins := true;
                  if courant < max_som
                   then if fact[courant+1] = 0 then fact[courant+1] := -1;
                  verif_coherence(courant,controle);
                  elem_som(courant);
                 end
       else begin
             if courant = 0 then courant := courant + 1;
             if fact[courant] = 0 then fact[courant] := 1;
             if courant < max_som
              then fact[courant + 1] := 1;
             elem_som(courant);
            end
  else begin
        signal_plus := true;
        if courant < max_som
         then if fact[courant+1] = 0 then fact[courant+1] := 1;
        verif_coherence(courant,controle);
        elem_som(courant);
       end;
 y('init_elem_som');
end;

(*********************************************************************)
(* procédure de reconnaissance globale d'une contrainte              *)
(* production du message d'erreur no 44 en cas de dépassement du     *)
(* nombre maximal de noms de contraintes permis                      *)
(*********************************************************************)

procedure contrainte;
 var cpt : integer;

 (*********************************************************************)
 (* procédure de reconnaissance du second membre d'une contrainte     *)
 (* production du message d'erreur no 46 en cas de dépassement du     *)
 (* nombre maximal d'éléments de somme permis                         *)
 (*********************************************************************)
```

```
procedure scd_mbre;
 begin
  x('scd_mbre');
  signal_plus := false;           (* initialisations *)
  signal_moins := false;
  signal_fois := false;
  sortie := false;          (* plus de problème de passage *)
  nb_ind_som := 0;          (* au second membre ici        *)
  elem_suiv;
  repeat
   init_elem_som(tab_equ[i].mult2,nb_ind_som,nb2_som[i],2);
  until (right(strw,1) = ':') or (right(strw,1) = ',');
  (* fin du paragraphe "equ:" ou autre contrainte *)
  if nb1_som[i] + nb2_som[i] > max_som then erreur(46);
  verif_coherence(nb2_som[i],2); (* vérification de cohérence du *)
  refvar := 0;                   (* dernier élément de somme du  *)
  refcst := 0;                   (* second membre et remise à 0  *)
  y('scd_mbre');
 end;

(*************************************************************************)
(* procédure de reconnaissance du premier membre d'une contrainte*)
(* production du message d'erreur no 22 en cas d'erreur de       *)
(* séparateur, no 46 en cas de dépassement du nombre maximal     *)
(* d'éléments de somme permis                                    *)
(*************************************************************************)

procedure prem_mbre;
 begin
  x('prem_mbre');
  ver_separateur(',',strw);
  if not ok then erreur(22);
  ver_nom(strw);                 (* vérification du nom *)
  ver_mot_res(strw);             (* de la contrainte    *)
  ver_fam_nom(strw,nb_fam);
  ver_nom_par(strw,nb_fam);
  ver_equ_nom(strw);
  ver1_var_nom(strw);
  place_nom(strw,tab_equ[i].nom);
  elem_suiv;
  nb_ind_som := 0;
  if strw[1] = '#'                (* quantification *)
   then begin
         ver_separateur(',',strw);
         if not ok then erreur(22);
         trt_ind_ptt(nb_ind_ptt,tab_equ[i].no_ent_ptt,
                     tab_equ[i].code_ptt,tab_equ[i].val_ptt,1);
        end;
  repeat
   init_elem_som(tab_equ[i].mult1,nb_ind_som,nb1_som[i],1);
  until (tab_equ[i].code_equ <> 0) or (sortie) or (right(strw,1) = '=');
  (* fin du premier membre *)
  if nb1_som[i] > max_som then erreur(46);
  verif_coherence(nb1_som[i],1); (* vérification de cohérence du *)
  refvar := 0;                   (* dernier élément de somme du  *)
  refcst := 0;                   (* premier membre et remise à 0 *)
```

```
   if (right(strw,1) = '=') and (tab_equ[i].code_equ = 0)
    then sep_sec;
   scd_mbre;
   y('prem-mbre');
  end;

 begin (* contrainte *)
  x('contrainte');
  i := 1;
  for cpt := 1 to max_equ do
   begin                              (* initialisations *)
    nb1_som[cpt] := 0;
    nb2_som[cpt] := 0;
   end;
  signal_plus := false;
  signal_moins := false;
  signal_fois := false;
  repeat
   prem_mbre;
   i := i + 1;
  until (i > max_equ) or (right(strw,1) = ':');
  if i > max_equ then erreur(44);
  nb_equ := i - 1;                   (* nombre total de contraintes *)
  nb_var := ivar - 1;                (* nombre total de variables   *)
  y('contrainte');
 end;

(*************************************************************)
(* procédure de reconnaissance de la fonction objectif *)
(* production du message d'erreur no 38 et no 16 en cas*)
(* d'erreur de séparateur, no 43 sur la nature de      *)
(* l'optimisation, no 46 en cas de dépassement du      *)
(* nombre maximal d'éléments de somme autorisé         *)
(*************************************************************)

procedure fonct_obj;
 var strwrk : str80;
 begin
  x('fo');
  i := 0;
  ivar := 1;
  refvar := 0;
  refcst := 0;
  sortie := false;          (* pas de problème de second membre ici *)
  ver_separateur('[',strw);
  if not ok then erreur(38);
  strwrk := left(strw,length(strw)-1);
  if strwrk <> 'min'
   then if strwrk <> 'max'
         then erreur(43)
         else fo.nature := false
   else fo.nature := true;
  signal_plus := false;                (* initialisations *)
  signal_moins := false;
  signal_fois := false;
  nb_ind_som := 0;
  nb_fo_som := 0;
  elem_suiv;
```

```
  repeat
   init_elem_som(fo.mult,nb_ind_som,nb_fo_som,0);
  until (nb_fo_som > max_som) or (right(strw,1) = ',')
        or (right(strw,1) = ']');
  (* fin de la fonction objectif ou début d'un premier membre de contrainte*)
  if nb_fo_som > max_som then erreur(46);
  verif_coherence(nb_fo_som,0); (* vérificaton de cohérence du dernier *)
  refcst := 0;                  (* élément de somme et remise à 0 des  *)
  refvar := 0;                  (* références                          *)
  if right(strw,1) = ']'
   then begin
         elem_suiv;
         if strw <> ';' then erreur(16);
         elem_suiv;
        end;
  y('fo');
 end;

(*************************************************************)
(* procédure d'initialisation de l'ensemble des variables *)
(*************************************************************)

procedure initialise;
 var cpt,cnt,tel : integer;
 begin
  x('initialise');
  ivar := 1;
  for cpt := 1 to max_var do
   begin
    tab_var[cpt].nom := '';
    for cnt := 1 to max_indices do tab_var[cpt].no_ent_par[cnt] := 0;
   end;
  for cpt := 1 to max_equ do
   begin
    tab_equ[cpt].nom := '';
    tab_equ[cpt].code_equ := 0;
    for cnt := 1 to max_indices do
     begin
      tab_equ[cpt].no_ent_ptt[cnt] := 0;
      tab_equ[cpt].code_ptt[cnt] := 0;
      tab_equ[cpt].val_ptt[cnt] := 0;
     end;
    for cnt := 1 to max_som do
     begin
      for tel := 1 to max_indices do
       begin
        tab_equ[cpt].no1_ent_som[cnt,tel] := 0;
        tab_equ[cpt].no2_ent_som[cnt,tel] := 0;
        tab_equ[cpt].code1_som[cnt,tel] := 0;
        tab_equ[cpt].code2_som[cnt,tel] := 0;
        tab_equ[cpt].val1_som[cnt,tel] := 0;
        tab_equ[cpt].val2_som[cnt,tel] := 0;
        tab_equ[cpt].code1_cst[cnt,tel] := 0;
        tab_equ[cpt].code2_cst[cnt,tel] := 0;
        tab_equ[cpt].val1_cst[cnt,tel] := 0;
        tab_equ[cpt].val2_cst[cnt,tel] := 0;
        tab_equ[cpt].code1_var[cnt,tel] := 0;
        tab_equ[cpt].val1_var[cnt,tel] := 0;
       end;
```

```
        tab_equ[cpt].cst1[cnt] := 0;
        tab_equ[cpt].cst2[cnt] := 0;
        tab_equ[cpt].var1[cnt] := 0;
        tab_equ[cpt].mult1[cnt] := 0;
        tab_equ[cpt].mult2[cnt] := 0;
       end;
     end;
    for cpt := 1 to max_som do
     begin
      for tel := 1 to max_indices do
       begin
        fo.no_ent_som[cpt,tel] := 0;
        fo.code_som[cpt,tel] := 0;
        fo.val_som[cpt,tel] := 0;
        fo.code_cst[cpt,tel] := 0;
        fo.val_cst[cpt,tel] := 0;
        fo.code_var[cpt,tel] := 0;
        fo.val_var[cpt,tel] := 0;
       end;
      fo.cst[cpt] := 0;
      fo.variable[cpt] := 0;
      fo.mult[cpt] := 0;
     end;
    for cpt := 1 to max_indices do
     begin
      fictif_code2_var[cpt] := 0;
      fictif_val2_var[cpt]:= 0;
      fictif_var2[cpt] := 0;
     end;
    y('initialise');
   end;

  begin (* intro_equ *)
   x('intro_equ');
   initialise;
   elem_suiv;
   fonct_obj;
   contrainte;
   y('intro_equ');
  end;

 overlay procedure intro_com;   (* overlay de niveau 2 (inutile mais...) *)
  begin                         (* lecture sans vérification jusqu'à     *)
   repeat                       (* l'obtention de "fin:"                 *)
    elem_suiv;
   until (right(strw,4) = 'fin:');
   strw := 'fin:';
  end;

  procedure init_memoire;    (* initialisations *)
   begin
    x('init memoire');
    debug1 := false;
    debug2 := false;
    init_options;
    init_elem_suiv;
```

```
   nb_fam := 0;
   nb_elem_fam := 0;
   nb_cst := 0;
   nb_elem_cst := 0;
   ok := true;
   y('init memoire');
  end;

(*************************************************************************)
(* procedures d'affichages des résultats de reconnaissance       *)
(* de syntaxe des différents paragraphes (familles, constantes, *)
(* equations) utilisées uniquement en période de tests           *)
(*************************************************************************)

(*
 procedure affiche_result_1;
  var i,j : integer;
      tb : table_index;
  begin
   ecrisln('');
   ecrisln(i2s(nb_fam,2) + ' familles :');
   for i := 1 to nb_fam do
    begin
     tb := tab_ind[i];
     ecrisln(i2s(i,3) + ' '+ tb.nom);
     if tb.num
      then begin
            ecrisln('par intervalle :');
            ecrisln('de ' + i2s(tb.borne_inf,5) + ' a ' + i2s(tb.borne_sup,5))
           end
      else begin
            ecrisln('par enumeration :');
            for j := tb.borne_inf to tb.borne_sup do
             ecris(mem_alpha[j] + ' ');
            ecrisln('');
           end;
     ecrisln('');
    end;
   attente;
  end;

 procedure affiche_result_2;
  var no,i,j : integer;
  begin
   ecrisln('');
   ecrisln(i2s(nb_cst,2) + ' constantes : ');
   for i := 1 to nb_cst do
    begin
     ecrisln(i2s(i,3) + ' ' + tab_cst[i].nom);
     ecris('  # : ');
     for j := 1 to max_indices do
      begin
       no := tab_cst[i].no_ent_ptt[j];
       ecris(i2s(j,3) + ' ');
       if no <> 0
        then begin
              ecris(tab_ind[no].nom);
             end;
```

```
      if j < max_indices then begin
                                ecris(',');
                               end;
     end;
    ecrisln('');
    ecris('fam : ');
    for j := 1 to max_indices do
     begin
      no := tab_cst[i].no_ent_par[j];
      ecris(i2s(j,3) + ' ');
      if no <> 0
       then begin
             ecris(tab_ind[no].nom);
            end;
      if j < max_indices then begin
                                ecris(',');
                               end;
     end;
    ecrisln('');
    no := tab_cst[i].debut;
    for j := 1 to tab_cst[i].total do
     begin
      ecris(r2s(mem_const[j + no - 1],8,4));
     end;
    ecrisln('');
   end;
  attente;
 end;

procedure affiche_result_3;
 var i,j,k : integer;
 begin
  ecrisln('');
  ecris('Fonction objectif : ');
  ecris('   Il faut ');
  if fo.nature then ecris('minimiser') else ecris('maximiser');
  ecris(' la fonction.');
  ecrisln('');
  ecrisln('no_elem ! no_ent_som   !      code_som      !      val_som');
  for i := 1 to nb_fo_som do
   begin
    ecris(i2s(i,3) + ' !');
    for j := 1 to 3 do
     ecris(i2s(fo.no_ent_som[i,j],5) + ' ');
    ecris('! ');
    for j := 1 to 3 do
     ecris(i2s(fo.code_som[i,j],5) + ' ');
    ecris('! ');
    for j := 1 to 3 do
     ecris(i2s(fo.val_som[i,j],5) + ' ');
    ecrisln('');
   end;
  ecrisln('');
  ecrisln('        !  cst          !      code_cst      !      val_cst');
```

```
for i := 1 to nb_fo_som do
 begin
  ecris(i2s(i,3) + '   ');
  ecris(i2s(fo.cst[i],5) + '                ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(fo.code_cst[i,j],5) + ' ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(fo.val_cst[i,j],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('          !  variable    !       code_var       !      val_var');
for i := 1 to nb_fo_som do
 begin
  ecris(i2s(i,3) + '   ');
  ecris(i2s(fo.variable[i],5) + '                ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(fo.code_var[i,j],5) + ' ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(fo.val_var[i,j],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('mult : ');
for i := 1 to nb_fo_som do
 ecrisln(i2s(i,3) + ' ' + r2s(fo.mult[i],8,3));
ecrisln('');
ecrisln('');
ecrisln('Equations :');
for i := 1 to nb_equ do
 begin
  ecrisln('   equation ' + i2s(i,3) + '  nom : "' + tab_equ[i].nom + '"');
  ecrisln('       no_ent_ptt         code_ptt           val_ptt');
  for j := 1 to 3 do
   ecris(i2s(tab_equ[i].no_ent_ptt[j],5) + ' ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(tab_equ[i].code_ptt[j],5) + ' ');
  ecris('! ');
  for j := 1 to 3 do
   ecris(i2s(tab_equ[i].val_ptt[j],5) + ' ');
  ecrisln('');
  ecrisln('Code = ' + i2s(tab_equ[i].code_equ,3));
  ecrisln('');
  ecrisln('1er membre : ');
  ecrisln('');
  ecrisln('          no1_ent_som    !     code1_som       !     val1_som');
  for j := 1 to nb1_som[i] do
   begin
    ecris(i2s(j,3) + '   ! ');
    for k := 1 to 3 do
     ecris(i2s(tab_equ[i].no1_ent_som[j,k],5) + ' ');
    ecris('! ');
    for k := 1 to 3 do
     ecris(i2s(tab_equ[i].code1_som[j,k],5) + ' ');
    ecris('! ');
```

```
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].val1_som[j,k],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('              cst1          !     code1_cst      !     val1_cst');
for j := 1 to nb1_som[i] do
 begin
  ecris(i2s(j,3) + '   ! ');
  ecris(i2s(tab_equ[i].cst1[j],5) + '                ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].code1_cst[j,k],5) + ' ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].val1_cst[j,k],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('              var1          !     code1_var      !     val1_var');
for j := 1 to nb1_som[i] do
 begin
  ecris(i2s(j,3) + '   ! ');
  ecris(i2s(tab_equ[i].var1[j],5) + '                ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].code1_var[j,k],5) + ' ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].val1_var[j,k],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('    mult1');
for j := 1 to nb1_som[i] do
 begin
  ecrisln(i2s(j,3) + ' ' + r2s(tab_equ[i].mult1[j],8,5));
 end;
 ecrisln('');
 ecrisln('2nd membre : ');
 ecrisln('');
 ecrisln('             no2_ent_som   !    code2_som      !     val2_som');
for j := 1 to nb2_som[i] do
 begin
  ecris(i2s(j,3) + '   ! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].no2_ent_som[j,k],5) + ' ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].code2_som[j,k],5) + ' ');
  ecris('! ');
  for k := 1 to 3 do
   ecris(i2s(tab_equ[i].val2_som[j,k],5) + ' ');
  ecrisln('');
 end;
ecrisln('');
ecrisln('              cst2          !     code2_cst      !     val2_cst');
```

```
   for j := 1 to nb2_som[i] do
    begin
     ecris(i2s(j,3) + '  ! ');
     ecris(i2s(tab_equ[i].cst2[j],5) + '                ');
     ecris('! ');
     for k := 1 to 3 do
      ecris(i2s(tab_equ[i].code2_cst[j,k],5) + ' ');
     ecris('! ');
     for k := 1 to 3 do
      ecris(i2s(tab_equ[i].val2_cst[j,k],5) + ' ');
     ecrisln('');
    end;
   ecrisln('');
   ecrisln('    mult2');
   for j := 1 to nb2_som[i] do
    begin
     ecrisln(i2s(j,3) + ' ' + r2s(tab_equ[i].mult2[j],8,5));
    end;
  ecrisln('');
  ecrisln('Variables :');
  for j := 1 to nb_var do
   begin
    ecris('"' + tab_var[j].nom + '"  ');
    for k := 1 to 3 do
     ecris(i2s(tab_var[j].no_ent_par[k],2) + ', ');
    ecrisln('');
   end;
  end;
  attente;
 end;
 *)

(*****************************************************************)
(* agencement du traitement de reconnaissance de la syntaxe *)
(*****************************************************************)

procedure compil;
 var fam_vu,cst_vu,equ_vu,com_vu : boolean;

 (*****************************************************************)
 (* procédure de reconnaissance de la syntaxe des familles et   *)
 (* de la cohérence des paragraphes à ce niveau                 *)
 (* production des messages no 35 et no 02 en cas d'incohérence *)
 (* dans l'apparition des paragraphes                           *)
 (*****************************************************************)

 procedure trt_fam;
  begin
   x('trt fam');
   if cst_vu or equ_vu or com_vu then erreur(35);
   if fam_vu then erreur(02);
   intro_index;
   msg('paragraphe "familles" traité',8,false);
   (*affiche_result_1;*)
   fam_vu := true;
   y('trt fam');
  end;
```

```
(*****************************************************************)
(* procédure de reconnaissance de la syntaxe des constantes et *)
(* de la cohérence des paragraphes à ce niveau                   *)
(* production des messages no 35 et no 02 en cas d'incohérence *)
(* dans l'apparition des paragraphes                             *)
(*****************************************************************)

procedure trt_cst;
 begin
  x('trt cst');
  if equ_vu or com_vu then erreur(35);
  if cst_vu then erreur(02);
  intro_cst;
  msg('paragraphe "constantes" traité',8,false);
  (* affiche_result_2; *)
  cst_vu := true;
  fam_vu := true;
  y('trt cst');
 end;

(*****************************************************************)
(* procédure de reconnaissance de la syntaxe des équations  et *)
(* de la cohérence des paragraphes à ce niveau                   *)
(* production du message no 02 en cas d'incohérence              *)
(* dans l'apparition des paragraphes                             *)
(*****************************************************************)

procedure trt_equ;
 begin
  x('trt equ');
  if equ_vu then erreur(02);
  intro_equ;
  msg('paragraphe "equations" traité',8,false);
  (* affiche_result_3; *)
  fam_vu := true;
  cst_vu := true;
  equ_vu := true;
  y('trt equ');
 end;

(*****************************************************************)
(* procédure de reconnaissance de la syntaxe des commentaires  *)
(* et de la cohérence des paragraphes à ce niveau                *)
(* production des messages no 02 et 34 en cas d'incohérence    *)
(* dans l'apparition des paragraphes                             *)
(*****************************************************************)

procedure trt_com;
 begin
  x('trt com');
  if not equ_vu then erreur(34);
  if com_vu then erreur(02);
  fam_vu := true;
  cst_vu := true;
  com_vu := true;
  intro_com;
  msg('paragraphe "commentaire" traité',8,false);
  y('trt com');
 end;
```

```
  (*******************************************************)
  (* production du message no 34 en cas d'incohérence *)
  (* dans l'apparition des paragraphes                   *)
  (*******************************************************)

  procedure trt_fin;
   begin
    x('trt fin');
    if not equ_vu then erreur(34);
    y('trt fin');
   end;

  begin (* compil *)
   x('compil');
   fam_vu := false;
   cst_vu := false;
   equ_vu := false;
   com_vu := false;
   elem_suiv;
   repeat
    if strw = 'fam:' then trt_fam else
    if strw = 'cst:' then trt_cst else
    if strw = 'equ:' then trt_equ else
    if strw = 'com:' then trt_com
    else erreur(03);
   until (strw = 'fin:');
   trt_fin;
   y('compil');
  end;

begin (* procedure compilation *)
 x('compilation');
 init_memoire;
 compil;
 close(f_in);
 y('compilation');
end;
```

```
(*****************************************************************)
(* procédure regroupant transformation de structures, simplexe, *)
(* analyse de sensibilité et paramétrage                         *)
(*****************************************************************)

overlay procedure calcul;  (* overlay de niveau 1 *)

 const max_li = 50;                (* nbre maximal  de lignes  *)
       max_col = 90;               (* nbre maximal de colonnes *)
       epsilon = 0.001;            (* seuil d'arrondi          *)
       maxreal = 9.999E+37;        (* approximation de ∞       *)
 type
      tab_simpl = array[1..max_li,1..max_col] of real;
      ent_equ = array[1..max_equ] of integer;
      ent_var = array[1..max_var] of integer;
      ent_col = array[1..max_col] of integer;
      bool_col = array[1..max_col] of boolean;
      reel_col = array[1..max_col] of real;
      ent_li = array[1..max_li] of integer;
      bool_li = array[1..max_li] of boolean;
      reel_li = array[1..max_li] of real;
      ptrsol = ^solution;
         (* record de mémorisation des résultats *)
         (* dans l'intervalle [inf,sup]          *)
      solution = record
                   inf,sup,
                   indep,pente : real; (* valeur de la fonction objectif     *
                                       (* partie (in)dépendante du paramètre *
                   col_sol : ent_li;   (* numéros de colonnes des variables  *
                   val_sol : reel_li;  (* en base et leurs valeurs           *
                   code1,code2 : integer; (* code relatif à la nature *)
                                          (* des résultats            *)
                   nb : real;
                   svt : ptrsol; (* pointeur vers le record suivant *)
                 end;
 var cpt,i,j,           (* compteurs de travail *)
     nb_col,nb_li,      (* nombre de colonnes et de lignes *)
     l_piv,col_piv,     (* no de ligne et de colonne du pivot *)
                        (* nbre de var d'écart de chaque signe*)
                        (* et nbre de variables artificielles *)
     nb_ve_plus,nb_ve_moins,nb_va : integer;
     z,pente,z_copie,pente_copie, (* valeur de la fonction objectif,   *)
                                  (* partie indépendante et dépendante *)
                                  (* du paramètre et leur copie        *)
     w,                           (* valeur de la fonction objectif du *)
                                  (* problème auxiliaire               *)
     lim_inf,lim_sup : real;      (* intervalle par défaut de validité *)
                                  (* de la solution                    *)
     base,                        (* numéros de colonnes où se trouve  *)
                                  (* la base de départ                 *)
     inverse_base : ent_li;       (* numéros de colonnes où se trouve  *)
                                  (* l'inverse de la base              *)
     c_base,                      (* coûts initiaux des variables en   *)
                                  (* base et dans le bon ordre         *)
     c_bis,                  (* copie de l'ensemble des coûts initiaux *)
     b,b_delta,              (* valeur des seconds membres des         *)
                             (* contraintes, partie (in)dépendante du  *)
     b_copie,b_delta_copie,  (* paramètre et leur copie                *)
     candi_li,               (* valeur des rapports candidats sortants *)
```

```
    b_lim_inf,b_lim_sup : reel_li; (* limites des intervalles pour le *)
                                   (* paramétrage sur les b           *)
    a,a_copie : tab_simpl;    (* tableau des seconds membres et copie  *)
    chge_fo : boolean;        (* booléen rendant compte de min ou max  *)
    place_equ,                (* nbre de lignes occupées par chaque    *)
                              (* contrainte (quantification)           *)
    ve,va : ent_equ;          (* mémorisation des contraintes où on    *)
                              (* trouve des var d'écart et/ou artif    *)
    place_var : ent_var;      (* nbre de colonnes occupées par chaque  *)
                              (* variable en fonction de ses indices   *)
    mem_un : ent_col;         (* pour chaque colonne de base, mémorisa-*)
                              (* tion de la ligne contenant le 1       *)
    d,                        (* coûts du problème auxiliaire          *)
    c,c_delta,c_copie,c_delta_copie, (* coûts, partie (in) dépendante *)
                                   (* du paramètre et leur copie    *)
    candi_col,              (* valeurs des rapports candidats entrants *)
    c_lim_inf,c_lim_sup : reel_col;(* limites des intervalles pour le *)
                                   (* paramétrage sur les c         *)
    ptr,premier,dernier : ptrsol;  (* pointeurs sur les records     *)
    s_anal_var,p_anal_var : bool_col; (* booléens de mémorisation des *)
    s_anal_li,p_anal_li : bool_li;    (* demandes d'analyse de sensi- *)
    c_par,b_par,                      (* bilité et paramétrage sur    *)
    c_sens,b_sens : boolean;          (* les b et les c               *)
    li_redond : bool_li;       (* mémorisation des lignes redondantes *)

(*********************************************************)
(* fonction réelle d'un paramètre réel renvoyant la  *)
(* valeur de ce paramètre arrondie à 0 s'il y a lieu *)
(*********************************************************)

function arrondi(r : real) : real;
 begin
  if abs(r) < epsilon
   then arrondi := 0.0
   else arrondi := r;
 end;

(*****************************************************************)
(* procédure de présentation et d'enregistrement des choix en *)
(* matière d'analyse de sensibilité et paramétrage           *)
(*****************************************************************)

procedure choix_analyse;
 var fini : boolean;
 begin
  clrscr;
  gotoxy(10,02); write('Désirez-vous :');
  gotoxy(17,04); write('Une analyse de sensibilité');
  gotoxy(23,06); write('sur les coûts                              (o/n) :');
  gotoxy(23,08); write('sur seconds membres des contraintes (o/n) :');
  (* ========== mis en réserve pour une utilisation ultérieure ========
  gotoxy(17,10); write('Un parametrage');
  gotoxy(23,12); write('sur les coûts                              (o/n) :');
  gotoxy(23,14); write('sur seconds membres des contraintes (o/n) :');
    ================================================================= *)
```

```
  c_sens := false;
  c_par  := false;
  b_sens := false;
  b_par  := false;
  repeat
   gotoxy(20,23); clreol;
   lis_boolean(67,06,c_sens);
   lis_boolean(67,08,b_sens);
   (* ========================
   lis_boolean(67,12,c_par);
   lis_boolean(67,14,b_par);
      ======================== *)
   gotoxy(20,18); write('Vos choix sont-ils définitifs : ');
   fini := true;
   lis_boolean(52,18,fini);
  until fini;
  clrscr;
 end;

(*************************************************************)
(* procédure d'affichage de la matrice des premiers membres  *)
(* des contraintes et des vecteurs associés, ainsi que Z et W *)
(*************************************************************)

(*
procedure affiche_result_4;
 var ld,cd,i,j : integer;
     r : real;
 begin
  if ecran
   then clrscr;
  ecrisln('   Résultats de la transformation');
  ecrisln('');
  ecrisln('Tableau "C" :');
  ecris('  ‖ ');
  for i := 1 to 10 do
   ecris(' ' + r2s(c[i],4,1));
  ecrisln('');
  ecrisln('');
  ecrisln('Tableau "D" :');
  ecris('  ‖ ');
  for i := 1 to 10 do
   ecris(' ' + r2s(d[i],4,1));
  ecrisln('');
  ecrisln('Tableaux "A" et "B"');
  for j := 1 to 10 do
   begin
    ecris(i2s(j,2) + ' ‖ ');
    for i := 1 to 10 do
     ecris(r2s(a[j,i],4,1) + ' ');
    ecrisln('  ‖  ' + r2s(b[j],4,1));
   end;
  ecrisln('');
  ecrisln('  z = ' + r2s(z,5,2));
  ecrisln('  w = ' + r2s(w,5,2));
  ecrisln('');
  attente;
 end;
 *)
```

```
(*****************************************************************)
(* procédure de compte-rendu des résultats pour le paramétrage *)
(*****************************************************************)

procedure non_real_bor;
 begin
  case ptr^.code1 of
   0 : ecrisln('pour des valeurs inférieures à ' + r2s(ptr^.nb,10,5) + ',');
   1 : ecrisln('pour des valeurs supérieures à ' + r2s(ptr^.nb,10,5) + ',');
  end;
  case ptr^.code2 of
   0 : erreur(76); (* pas de solutions réalisables *)
   1 : erreur(77); (* solutions optimales non bornées *)
  end;
 end;

 (*****************************************************************)
 (* fonction booléenne renvoyant la valeur VRAI si la colonne no *)
 (* NO_COL est une colonne de base (vecteurs associés compris)   *)
 (*****************************************************************)

 function col_base(no_col : integer) : boolean;
 var cpt,cpt_un : integer;
     cb_res : boolean;
 begin
  x('col_base');
  cpt_un := 0;
  cb_res := true;
  if (d[no_col] = 0) and (c[no_col] = 0)
   then begin
         for cpt := 1 to nb_li do
          begin
           if a[cpt,no_col] = 1
            then begin
                  cpt_un := cpt_un + 1;
                  mem_un[no_col] := cpt;
                 end
            else if a[cpt,no_col] <> 0
                  then cb_res := false;
          end;
          if cpt_un <> 1 then cb_res := false;
        end
   else cb_res := false;
  col_base := cb_res;
  y('col_base');
 end;

(*****************************************************************)
(* procédure déterminant le nombre d'éléments NB1, NB2 ,NB3 des *)
(* familles reprises dans TAB sous les références REF1, REF2,   *)
(* REF3 en tenant compte des particularités enregistrées dans   *)
(* TABCODE si la valeur de CODE l'exige                         *)
(*****************************************************************)
```

```
procedure fastidieuse(tab,tabcode : tableau;
                      var ref1,ref2,ref3,nb1,nb2,nb3 : integer;
                      code : integer);
 begin
  x('fastidieuse');
  ref1 := tab[1];
  ref2 := tab[2];
  ref3 := tab[3];
  if ref1 <> 0
   then if code = 1
         then if tabcode[1] <> 0
               then nb1 := largeur(ref1) - 1
               else nb1 := largeur(ref1)
         else nb1 := largeur(ref1)
   else nb1 := 1;
  if ref2 <> 0
   then if code = 1
         then if tabcode[2] <> 0
               then nb2 := largeur(ref2) - 1
               else nb2 := largeur(ref2)
         else nb2 := largeur(ref2)
   else nb2 := 1;
  if ref3 <> 0
   then if code = 1
         then if tabcode[3] <> 0
               then nb3 := largeur(ref3) - 1
               else nb3 := largeur(ref3)
         else nb3 := largeur(ref3)
   else nb3 := 1;
  y('fastidieuse');
 end;

(*************************************************************************)
(* procédure renvoyant dans RENDRE le numéro de l'élément minimal        *)
(* parmi les NB premiers du tableau CANDI_LI                             *)
(*************************************************************************)

procedure li_det_min(nb : integer;
                     var rendre : integer);
 var min : real;
     cnt : integer;
 begin
  x('li_det_min');
  rendre := 1;
  min := candi_li[1];
  for cnt := 2 to nb do
   if candi_li[cnt] < min
    then begin
          min := candi_li[cnt];
          rendre := cnt;
         end;
  y('li_det_min');
 end;

(*************************************************************************)
(* procédure renvoyant dans RENDRE le numéro de l'élément minimal        *)
(* parmi les NB premiers du tableau CANDI_COL                            *)
(*************************************************************************)
```

```
procedure col_det_min(nb : integer;
                      var rendre : integer);
 var min : real;
     cnt : integer;
 begin
  x('col_det_min');
  rendre := 1;
  min := candi_col[1];
  for cnt := 2 to nb do
   begin
    if candi_col[cnt] < min
     then begin
           min := candi_col[cnt];
           rendre := cnt;
          end;
   end;
  y('col_det_min');
 end;

(*************************************************************)
(* procédure de pivotage autour de l'élément se trouvant à la   *)
(* ligne LI et à la colonne COL                                 *)
(* en fonction de la valeur de NB, le pivotage est effectué sur *)
(* différents vecteurs associés                                 *)
(*************************************************************)

procedure pivotage(li,col,nb : integer);
 var retenir,d_retenir,c_retenir,delta_retenir : real;
 begin
  x('pivotage');
  ecrisln('pivotage autour de ' + i2s(li,2) + ' et ' + i2s(col,2));
  ecrisln('');
  b[li] := arrondi(b[li]/a[li,col]);              (* ligne pivot / pivot  *)
  retenir := a[li,col];
  for j := 1 to nb_col do a[li,j] := arrondi(a[li,j]/retenir);
  a[li,col] := 1;                                 (* nec pour exactitude  *)
  z := arrondi(z - c[col] * b[li]);
  case nb of
   2 : w := arrondi(w - d[col] * b[li]);
   3 : pente := arrondi(pente - c[col] * b_delta[li]);
   4 : pente := arrondi(pente - c_delta[col] * a[li,j]);
  end;
  for i := 1 to nb_li do
   if (i <> li) and (not li_redond[i])
    then begin
          b[i] := arrondi(b[i] - b[li] * a[i,col]);
          if nb = 3 then b_delta[i] := arrondi(b_delta[i] -
                                              b_delta[li] * a[i,col]);
          retenir := a[i,col];
          for j := 1 to nb_col do a[i,j] := arrondi(a[i,j] -
                                              a[li,j] * retenir);
         end;
  c_retenir := c[col];
  if nb = 2 then d_retenir := d[col];
  if nb = 4 then delta_retenir := c_delta[col];
```

```
  for j := 1 to nb_col do
   begin
    c[j] := arrondi(c[j] - c_retenir * a[li,j]);
    if nb = 2 then d[j] := arrondi(d[j] - d_retenir * a[li,j]);
    if nb = 4 then c_delta[j] := arrondi(c_delta[j] -
                                        delta_retenir * a[li,j]);
   end;
  c[col] := 0;
  if nb = 2 then d[col] := 0;
  if nb = 4 then c_delta[col] := 0;
  (* affiche_result_4; *)
  y('pivotage');
 end;

(*****************************************************************)
(* recherche de la variable sortante pour l'algorithme primal *)
(*****************************************************************)

procedure prim_rech_sort;
 var i,no,mem : integer;
 begin
  x('prim_rech_sort');
  i := 1;
  no := 0;
  repeat
   if (a[i,col_piv] > 0) and (not li_redond[i])
    then begin
           no := no + 1;
           candi_li[i] := b[i]/a[i,col_piv];
           mem := i;
         end
    else candi_li[i] := maxreal;
   i := i + 1;
  until i > nb_li;
  case no of
   0 : erreur(77);
   1 : l_piv := mem;
   else li_det_min(mem,l_piv);
  end;
  y('prim_rech_sort');
 end;

(*************************************************)
(* recherche de la variable entrante pour le dual *)
(*************************************************)

procedure dual_rech_entr;
 var j,no,mem : integer;
 begin
  x('dual_rech_entr');
  j := 1;
  no := 0;
```

```
        else begin
               equ_nom[no_ligne] := equnom + mem_alpha[i] + ')';
               no_ligne := no_ligne + 1;
              end;
      end;
 end;
y('gen 1');
nd;

rocedure gen_2;
ar i,j : integer;
   equnom : string[44];
   snum : string[10];
egin
 x('gen 2');
 for i := tab_ind[ptt[1]].borne_inf to tab_ind[ptt[1]].borne_sup do
  begin
   if not trou[1] or (i <> hole[1])
    then begin
          for j := tab_ind[ptt[2]].borne_inf to
                   tab_ind[ptt[2]].borne_sup do

           begin
            if not trou[2] or (j <> hole[2])
             then begin
                   equnom :=  tab_equ[cpt].nom + '(';
                   if tab_ind[ptt[1]].num
                    then begin
                          str(i,snum);
                          equnom := equnom + snum + ',';
                         end
                    else begin
                          equnom := equnom + mem_alpha[i] + ',';
                         end;
                   if tab_ind[ptt[2]].num
                    then begin
                          str(j,snum);
                          equ_nom[no_ligne] := equnom + snum + ')';
                          no_ligne := no_ligne + 1;
                         end
                    else begin
                          equ_nom[no_ligne] := equnom + mem_alpha[j] + ')';
                          no_ligne := no_ligne + 1;
                         end;

                  end;

             end;
           end;

    end;
   y('gen 2');
  end;

 procedure gen_3;
  var i,j,k : integer;
      equnom : string[44];
      snum : string[10];
```

```
alpha[i] + ')';

1]].borne_sup do

';

- ','

.pha[i] + ',';

om + snum + ')';
;

om + mem_alpha[j] + ')';
;
```

```
begin
 x('gen 3');
 for i := tab_ind[ptt[1]].borne_inf to tab_ind[pt
  begin
   if not trou[1] or (i <> hole[1])
    then begin
          for j := tab_ind[ptt[2]].borne_inf to
                   tab_ind[ptt[2]].borne_sup do
           begin
            if not trou[2] or (j <> hole[2])
             then begin
                   for k := tab_ind[ptt[3]].borne
                            tab_ind[ptt[3]].borne
                     begin
                      if not trou[3] or (k <> hol
                        then begin
                              equnom :=  tab_equ[
                              if tab_ind[ptt[1]].
                               then begin
                                     str(i,snum);
                                     equnom := eq
                                    end
                               else begin
                                     equnom := eq
                                    end;
                              if tab_ind[ptt[2]]
                               then begin
                                     str(j,snum)
                                     equnom := eq
                                    end
                               else begin
                                     equnom := eq
                                    end;
                              if tab_ind[ptt[3]]
                               then begin
                                     str(k,snum)
                                     equ_nom[no_

                                     no_ligne :=
                                    end
                               else begin
                                     equ_nom[no_

                                     no_ligne :=
                                    end;
                            end;
                     end;
                  end;
              end;
            end;
          end;
  end;
 y('gen 3');
end;
```

```
_ind[ptt[1]].borne_sup do


                                    s do
nf to
up do                               .no_ent_ptt[cnt];


])
                                    e;
].borne_inf to                      .code_ptt[cnt] of
].borne_sup do


 <> hole[3])                        := true;
                                    := tab_equ[cpt].val_ptt[cnt] -
ab_equ[cpt].nom + '(';                 tab_ind[ptt[cnt]].borne_inf;
:t[1]].num


 snum);
 := equnom + snum + ',';


                                    2] =  0) and (ptt[3] =  0) then gen_0;
                                    2] =  0) and (ptt[3] =  0) then gen_1;
 := equnom + mem_alpha[i] + ',';    2] <> 0) and (ptt[3] =  0) then gen_2;
                                    2] <> 0) and (ptt[3] <> 0) then gen_3;
 t[2]].num


 snum);
 := equnom + snum + ',';
                                    ******************************************)
                                    pour chaque variable du problème        *)
  := equnom + mem_alpha[j] + ',';   de la variable suivi le cas échéant de  *)
                                    valeurs permises pour les différentes   *)
 t[3]].num                          la variable, séparées par des virgules  *)
                                    ******************************************)

 snum);
m[no_ligne] := equnom + snum +
                ')';                3 : integer;
ne := no_ligne + 1;


m[no_ligne] := equnom +             [cpt].nom;
               mem_alpha[j] + ')';
ne := no_ligne + 1;




                                    ne_inf to tab_ind[ref1].borne_sup do


                                    = tab_var[cpt].nom + '(' + strwrk + ')';
                                    - 1;
```

```
    else begin
          var_nom[no_col] := tab_var[cpt].nom + '(' + mem_alpha[i] + ')';
          no_col := no_col + 1;
         end;
end;

procedure val2;
 var i,j : integer;
     str1,str2 : string[10];
 begin
  for i := tab_ind[ref1].borne_inf to tab_ind[ref1].borne_sup do
   for j := tab_ind[ref2].borne_inf to tab_ind[ref2].borne_sup do
    begin
     if tab_ind[ref1].num
      then begin
            str(i,str1);
            var_nom[no_col] := tab_var[cpt].nom + '(' + str1 + ',';
           end
      else begin
            var_nom[no_col] := tab_var[cpt].nom + '(' + mem_alpha[i] + ',';
           end;
     if tab_ind[ref2].num
      then begin
            str(j,str2);
            var_nom[no_col] := var_nom[no_col] + str2 + ')';
            no_col := no_col + 1;
           end
      else begin
            var_nom[no_col] := var_nom[no_col] + mem_alpha[j] + ')';
            no_col := no_col + 1;
           end;
    end;
 end;

procedure val3;
 var i,j,k : integer;
     str1,str2,str3 : string[10];
 begin
  for i := tab_ind[ref1].borne_inf to tab_ind[ref1].borne_sup do
   for j := tab_ind[ref2].borne_inf to tab_ind[ref2].borne_sup do
    for k := tab_ind[ref3].borne_inf to tab_ind[ref3].borne_sup do
     begin
      if tab_ind[ref1].num
       then begin
             str(i,str1);
             var_nom[no_col] := tab_var[cpt].nom + '(' + str1 + ',';
            end
       else begin
             var_nom[no_col] := tab_var[cpt].nom + '(' + mem_alpha[i] + '
            end;
      if tab_ind[ref2].num
       then begin
             str(j,str2);
             var_nom[no_col] := var_nom[no_col] + str2 + ',';
            end
       else begin
             var_nom[no_col] := var_nom[no_col] + mem_alpha[j] + ',';
            end;
```

```
        if tab_ind[ref3].num
         then begin
               str(k,str3);
               var_nom[no_col] := var_nom[no_col] + str3 + ')';
               no_col := no_col + 1;
              end
         else begin
               var_nom[no_col] := var_nom[no_col] + mem_alpha[k] + ')';
               no_col := no_col + 1;
              end;
       end;
   end;

 begin (* genere_nom_var *)
  x('gen nom var');
  no_col := 1;
  for cpt := 1 to nb_var do
   begin
    ref1 := tab_var[cpt].no_ent_par[1];
    ref2 := tab_var[cpt].no_ent_par[2];
    ref3 := tab_var[cpt].no_ent_par[3];
    if (ref1 <> 0) and (ref2 <> 0) and (ref3 <> 0) then val3;
    if (ref1 <> 0) and (ref2 <> 0) and (ref3 =  0) then val2;
    if (ref1 <> 0) and (ref2 =  0) and (ref3 =  0) then val1;
    if (ref1 =  0) and (ref2 =  0) and (ref3 =  0) then val0;
   end;
  y('gen nom var');
 end;

(*************************************************************************)
(* fonction entière renvoyant le nombre de colonnes occupées par *)
(* l'ensemble des variables proprement dites du problème         *)
(*************************************************************************)

 function equ_colonne : integer;
 var cpt,col_prise : integer;
 begin
  col_prise := 0;
  for cpt := 1 to nb_var do
   col_prise := col_prise + place_var[cpt];
  equ_colonne := col_prise;
 end;

(*************************************************************************)
(* procedure générant un nom pour chaque variable d'écart introduite  *)
(* automatiquement dans le problème pour sa résolution                *)
(* ce nom est le nom du numéro de ligne correspondant suivi de "ve+"  *)
(* ou "ve-" selon le signe de la variable introduite                  *)
(*************************************************************************)

procedure ve_nom;
 var cpt,cnt,li_prise,controle,col : integer;
```

```
  begin
   for cpt := 1 to nb_equ do
    if ve[cpt] > 0
     then begin
           li_prise := 0;
           for cnt := 1 to cpt-1 do
             li_prise := li_prise + place_equ[cnt];
           col := equ_colonne + li_prise;
           for controle := 1 to place_equ[cpt] do
            var_nom[col + controle] := equ_nom[li_prise + controle]
                                              + ',ve+';
          end
     else if ve[cpt] < 0
           then begin
                 li_prise := 0;
                 for cnt := 1 to cpt-1 do li_prise := li_prise + place_equ[cn
                 col := equ_colonne + nb_ve_plus + li_prise;
                 for controle := 1 to place_equ[cpt] do
                 var_nom[col + controle] := equ_nom[li_prise + controle]
                                                  + ',ve-';
                end;
  end;

  (*************************************************************************)
  (* procedure générant un nom pour chaque variable artificielle           *)
  (* introduite automatiquement dans le problème pour sa résolution        *)
  (* ce nom est le nom du numéro de ligne correspondant suivi de "va"      *)
  (*************************************************************************)

  procedure va_nom;
   var cpt,cnt,li_prise,controle,col : integer;
   begin
    for cpt := 1 to nb_equ do
     if va[cpt] <> 0
      then begin
           li_prise := 0;
           for cnt := 1 to cpt-1 do li_prise := li_prise + place_equ[cnt];
           col := equ_colonne + nb_ve_plus + nb_ve_moins + li_prise;
           for controle := 1 to place_equ[cpt] do
            var_nom[col + controle] := equ_nom[li_prise + controle]
                                              + ',va';
          end;
  end;

 (*************************************************************************)
 (* procédure regroupant transformation des structures et simplexe *)
 (* production du message d'erreur no 72 en cas de dépassement de  *)
 (* taille des structures du simplexe                              *)
 (*************************************************************************)

 overlay procedure calcul_et_simplexe;  (* overlay de niveau 2 *)
```

```
(*****************************)
(* échange de deux entiers *)
(*****************************)

procedure ech_ent(var xx,yy : integer);
 var ech : integer;
 begin
  x('ech ent');
  ech := xx;
  xx := yy;
  yy := ech;
  y('ech ent');
 end;

(*****************************)
(* échange de deux réels   *)
(*****************************)

procedure ech_reel(var xx,yy : real);
 var ech : real;
 begin
  x('ech reel');
  ech := xx;
  xx := yy;
  yy := ech;
  y('ech reel');
 end;

(*****************************************************************)
(* tri par valeur croissante sur les NB_ELEM références aux *)
(* variables de l'équation no NO_EQU                             *)
(* méthode tri bulle                                             *)
(*****************************************************************)

procedure tri(nb_elem,no_equ : integer);
 var cpt : integer;
     fini : boolean;

 (*******************************************)
 (* échange de tous les éléments nécessaires *)
 (* lorsque cela se justifie                 *)
 (*******************************************)

 procedure passe;
  var tel : integer;
  begin
  x('passe');
   while cpt < nb_elem do
    begin
     if tab_equ[no_equ].var1[cpt] > tab_equ[no_equ].var1[cpt+1]
      then begin
            ech_ent(tab_equ[no_equ].var1[cpt],tab_equ[no_equ].var1[cpt+1]);
            ech_ent(tab_equ[no_equ].cst1[cpt],tab_equ[no_equ].cst1[cpt+1]);
            ech_reel(tab_equ[no_equ].mult1[cpt],tab_equ[no_equ].mult1[cpt+1
```

```
            for tel := 1 to max_indices do
             begin
              ech_ent(tab_equ[no_equ].no1_ent_som[cpt,tel],
                      tab_equ[no_equ].no1_ent_som[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].code1_som[cpt,tel],
                      tab_equ[no_equ].code1_som[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].val1_som[cpt,tel],
                      tab_equ[no_equ].val1_som[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].code1_cst[cpt,tel],
                      tab_equ[no_equ].code1_cst[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].val1_cst[cpt,tel],
                      tab_equ[no_equ].val1_cst[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].code1_var[cpt,tel],
                      tab_equ[no_equ].code1_var[cpt+1,tel]);
              ech_ent(tab_equ[no_equ].val1_var[cpt,tel],
                      tab_equ[no_equ].val1_var[cpt+1,tel]);
             end;
            fini := false;
           end;
      cpt := cpt + 2;
     end;
     y('passe');
   end;

 begin (* tri *)
  x('tri');
  if odd(nb_elem) then begin
                        nb_elem := nb_elem + 1;
                        tab_equ[no_equ].var1[nb_elem] := maxint;
                       end;
  repeat
   fini := true;
   cpt := 1;
   passe;
   cpt := 2;
   passe;
  until fini;
  y('tri');
 end;

(*********************************************************)
(* même travail que ci-dessus pour la fonction objectif *)
(*********************************************************)

procedure tri_fo;
 var bis_nb,cpt : integer;
     fini : boolean;

 procedure passe_fo;
  var tel : integer;
  begin
  x('passe fo');
   while cpt < bis_nb do
    begin
     if fo.variable[cpt] > fo.variable[cpt+1]
      then begin
            ech_ent(fo.variable[cpt],fo.variable[cpt+1]);
            ech_ent(fo.cst[cpt],fo.cst[cpt+1]);
            ech_reel(fo.mult[cpt],fo.mult[cpt+1]);
```

```
            for tel := 1 to max_indices do
             begin
              ech_ent(fo.no_ent_som[cpt,tel],
                      fo.no_ent_som[cpt+1,tel]);
              ech_ent(fo.code_som[cpt,tel],
                      fo.code_som[cpt+1,tel]);
              ech_ent(fo.val_som[cpt,tel],
                      fo.val_som[cpt+1,tel]);
              ech_ent(fo.code_cst[cpt,tel],
                      fo.code_cst[cpt+1,tel]);
              ech_ent(fo.val_cst[cpt,tel],
                      fo.val_cst[cpt+1,tel]);
              ech_ent(fo.code_var[cpt,tel],
                      fo.code_var[cpt+1,tel]);
              ech_ent(fo.val_var[cpt,tel],
                      fo.val_var[cpt+1,tel]);
             end;
            fini := false;
           end;
     cpt := cpt + 2;
    end;
    y('passe fo');
 end;

begin (* tri_fo *)
 bis_nb := nb_fo_som;
 if odd(bis_nb) then begin
                      bis_nb := nb_fo_som + 1;
                      fo.variable[bis_nb] := maxint;
                     end;
 repeat
  fini := true;
  cpt := 1;
  passe_fo;
  cpt := 2;
  passe_fo;
 until fini;
end;

(*************************************************************)
(* détermination du nbre de colonnes nécessaires pour les *)
(* variables apparaissant dans la fonction objectif et les*)
(* contraintes, compte-tenu de leurs indices                 *)
(* production du message no 72  en cas de dépassement de  *)
(* valeur maximale permise                                   *)
(*************************************************************)

procedure det_place_col;
 var cpt,mem_place,ref1,ref2,ref3,nb1,nb2,nb3 : integer;
 begin
  x('det_place_col');
  nb_col := 0;
  for cpt := 1 to nb_var do
   begin
    fastidieuse(tab_var[cpt].no_ent_par,code_fictif,
                ref1,ref2,ref3,nb1,nb2,nb3,2);
    place_var[cpt] := nb1 * nb2 * nb3;
    nb_col := nb_col + place_var[cpt];
   end;
```

```
  if nb_col > max_col then erreur(72);
  y('det_place_col');
 end;

(*************************************************************)
(* détermination du nbre de lignes  nécessaires pour les  *)
(* contraintes, compte-tenu de la quantification          *)
(* production du message no 72  en cas de dépassement de  *)
(* valeur maximale permise                                *)
(*************************************************************)

procedure det_place_li;
 var cpt : integer;
 begin
 x('det_place_li');
  nb_li := 0;
  for cpt := 1 to nb_equ do
   nb_li := nb_li + place_equ[cpt];
  if nb_li > max_li then erreur(72);
  y('det_place_li');
 end;

(*************************************************************************)
(* fonction entière renvoyant un numéro égal à la position occupée   *)
(* par un élément (variable ou constante) comportant deux familles   *)
(* reprises dans TAB et dont l'état actuel est resp. I et J,         *)
(* position calculée par déplacement relatif à la position occupée   *)
(* par le premier élément de ce type (premier état de chaque famille)*)
(* exemple : si I = [-2_2], on aura les états 0,1,2,3 et 4           *)
(*           si VILLE = (LLN,BXL,NAM) on aura les états 0,1 et 2     *)
(*************************************************************************)

function deux(i,j : integer;tab : tableau) : integer;
 var ref1,ref2,l2 : integer;
 begin
 x('f 2');
  ref1 := tab[1];
  ref2 := tab[2];
  l2 := largeur(ref2);
  deux := (i - 1) * l2 + j;
  y('f 2');
 end;

(*************************************************************************)
(* même rôle que la procédure précédente pour un élément à 3 familles *)
(*************************************************************************)

function trois(i,j,k : integer;tab : tableau) : integer;
 var ref1,ref2,ref3,l2,l3 : integer;
 begin
 x('f 3');
  ref1 := tab[1];
  ref2 := tab[2];
  ref3 := tab[3];
  l2 := largeur(ref2);
  l3 := largeur(ref3);
  trois := (i - 1) * l2 * l3 + (j - 1) * l3 + k;
  y('f 3');
 end;
```

```
(*************************************************************************)
(* fonction entière renvoyant le numéro de la colonne occupée par la     *)
(* variable no REF et dont l'état actuel des familles est resp. I, J, K *)
(*************************************************************************)

function colonne(ref,i,j,k : integer) : integer;
 var col_prise,cpt : integer;
begin
  x('colonne');
  col_prise := 0;
  for cpt := 1 to ref-1 do col_prise := col_prise + place_var[cpt];
  if (i = 0) and (j = 0) and (k = 0)
   then colonne := col_prise + 1
   else if ((i <> 0) and (j = 0) and (k = 0))
          or ((i = 0) and (j <> 0) and (k = 0))
          or ((i = 0) and (j = 0) and (k <> 0))
         then colonne := col_prise + i + j + k
         else if (i <> 0) and (j <> 0) and (k = 0)
                then colonne :=  col_prise + deux(i,j,tab_var[ref].no_ent_par
                else if (i <> 0) and (j = 0) and (k <> 0)
                       then colonne := col_prise +
                                       deux(i,k,tab_var[ref].no_ent_par)
                       else if (i = 0) and (j <> 0) and (k <> 0)
                              then colonne := col_prise +
                                              deux(j,k,tab_var[ref].no_ent_par)
                              else if (i <> 0) and (j <> 0) and (k <> 0)
                                     then colonne := col_prise +
                                                     trois(i,j,k,tab_var[ref].no_ent_par)
  y('colonne');
  end;

(*************************************************************************)
(* fonction réelle renvoyant la valeur de la constante no REF dont *)
(* l'état actuel des familles est respectivement I, J, K           *)
(*************************************************************************)

function val_cst(ref,i,j,k : integer) : real;
 var no : integer;
 begin
  x('val cst');
  if (i = 0) and ( j = 0) and (k = 0)
   then no := tab_cst[ref].debut
   else if ((i <> 0) and (j = 0) and (k = 0))
          or ((i = 0) and (j <> 0) and (k = 0))
          or ((i = 0) and (j = 0) and (k <> 0))
         then no := tab_cst[ref].debut + i + j + k - 1
         else if (i <> 0) and (j <> 0) and (k = 0)
                then no := tab_cst[ref].debut +
                           deux(i,j,tab_cst[ref].no_ent_par) - 1
                else if (i <> 0) and (j = 0) and (k <> 0)
                       then no := tab_cst[ref].debut +
                                  deux(i,k,tab_cst[ref].no_ent_par) - 1
                       else if (i = 0) and (j <> 0) and (k <> 0)
                              then no := tab_cst[ref].debut +
                                         deux(j,k,tab_cst[ref].no_ent_par) - 1
                              else if (i <> 0) and (j <> 0) and (k <> 0)
                                     then no := tab_cst[ref].debut +
                                                trois(i,j,k,tab_cst[ref].no_ent_par) - 1;
```

```
  val_cst := mem_const[no];
  y('val cst');
 end;

(*************************************************************************)
(* procédure réalisant la transformation de structures entre structures *)
(* générées par la reconnaissance de la syntaxe et structures de la     *)
(* méthode du simplexe                                                   *)
(*************************************************************************)

procedure compl_init;
var no,
    p1,p2,p3,s1,s2,s3, (* compteurs d'états sur les # et les SOM *)
    refcst,refvar,no_ligne : integer;
    ptt,som,           (* références aux familles de # et de sommation    *
    maxp,maxs,         (* nombre maximum -1 d'états pour les # et les som *
    exp,exs : tableau; (* mémorisation de l'état pour lequel il y a <>    *
    mult : real;


 (*************************************************************************
 (* fonction entière renvoyant l'état courant de la famille référencée NB,
 (* état courant d'une famille de variable ou de constante, déterminé à
 (* partir de l'etat de la famille correspondante dans les sommes et les #
 (*************************************************************************

 function ind_boucle(nb : integer) : integer;
  var i_b : integer;
  begin
   x('ind boucle');
   if (nb = ptt[1]) and (maxp[1] <> 0)  (* plusieurs valeurs de quantif *)
    then i_b := p1 + 1
    else if (nb = ptt[2]) and (maxp[2] <> 0)
          then i_b := p2 + 1
          else if (nb = ptt[3]) and (maxp[3] <> 0)
                then i_b := p3 + 1
                else if (nb = som[1]) and (maxs[1] <> 0)
                      then i_b := s1 + 1
                      else if (nb = som[2]) and (maxs[2] <> 0)
                            then i_b := s2 + 1
                            else if (nb = som[3]) and (maxs[3] <> 0)
                                  then i_b := s3 + 1
                                  else i_b := 0;
  ind_boucle := i_b;
  y('ind boucle');
 end;

(*************************************************************************)
(* procédure recherchant dans TABCODE et TABVAL les éventuelles          *)
(* particularités des noms de familles référencés dans TAB et mémorisant*)
(* dans INDICE les états des particularités rencontrées                  *)
(* Production du message d'erreur no 70 en cas d'erreur sur les valeurs *)
(* prises par les indices                                                *)
(*************************************************************************)
```

```
procedure cherche_cas_special(var tab,tabcode,tabval,indice : tableau);
 var i,max : integer;
 begin
  x('cherche_cas_special');
  for i := 1 to max_indices do
   case tabcode[i] of
   0   : ;
   1,2 : begin (* valeur particulière *)
          indice[i] := tabval[i] - tab_ind[tab[i]].borne_inf + 1;
                              (* numéro d'énumération *)
         end;
   3   : begin (* décalage *)
          max := tab_ind[tab[i]].borne_sup - tab_ind[tab[i]].borne_inf;
          indice[i] := indice[i] + tab_ind[tabval[i]].borne_inf
                                 - tab_ind[tab[i]].borne_inf;
          if (indice[i] < 0) or (indice[i] > max) then erreur(70);
         end;
   end;
  y('cherche_cas_special');
  end;

(*************************************************************************)
(* initialisation des boucles sur les références aux familles reprises  *)
(* dans TAB, en tenant compte des particularités enregistrées dans      *)
(* TABCODE et TABVAL                                                     *)
(*************************************************************************)

procedure pour_som(var tab,tabcode,tabval : tableau);
 var cnt : integer;
 begin
  x('pour som');
  for cnt := 1 to max_indices do
   begin
    som[cnt] := tab[cnt];
    exs[cnt] := - 1;
    if som[cnt] = 0
     then maxs[cnt] := 0
     else begin
           maxs[cnt] := largeur(som[cnt]) - 1;
           case tabcode[cnt] of
           0,3 : ; (* pas de trou    *)
           1,2 : exs[cnt] := tabval[cnt] - tab_ind[tab[cnt]].borne_inf;
                              (* no énumération *)

           end;
          end;
   end;
   y('pour som');
 end;

(*************************************************************************)
(* procédure permettant l'éclatement de la contrainte no NO_EQU en      *)
(* fonction de la quantification et de ses éventuelles particularités *)
(* CODE indique si la procédure contribue à initialiser la matrice des*)
(* premiers membres, le vecteur des seconds membres ou les coûts       *)
(*************************************************************************)

 procedure boucles_ptt(code,no_equ : integer);
  var valeur : real;
      cpt_ele,no_col,i : integer;
      ind_var,ind_cst : tableau;
```

```
(*********************************************************************)
(* remplissage de la matrice des premiers membres des contraintes *)
(*********************************************************************)

procedure pour_a;
 begin
  x('pour a');
  no_col := colonne(refvar,ind_var[1],ind_var[2],ind_var[3]);
  a[no_ligne,no_col] := a[no_ligne,no_col] + valeur;
  y('pour a');
 end;

(*********************************************************************)
(* remplissage du vecteur des seconds  membres des contraintes *)
(*********************************************************************)

procedure pour_b;
 begin
  x('pour b');
  b[no_ligne] := b[no_ligne] + valeur;
  y('pour b');
 end;

(*********************************************************************)
(* remplissage du vecteur des coûts de la fonction objectif *)
(*********************************************************************)

procedure pour_c;
 begin
  x('pour c');
  if refvar <> 0
   then begin
         no_col := colonne(refvar,ind_var[1],ind_var[2],ind_var[3]);
         c[no_col] := c[no_col] + valeur;
        end
   else z := z - mult;
 y('pour c');
 end;


(*********************************************************************)
(* procédure initialisant et effectuant les boucles sur les familles *)
(* de sommation pour la matrice des premiers membres                 *)
(*********************************************************************)

procedure boucles_som;
 begin
  x('boucles_som');
  for s1 := 0 to maxs[1] do  (* boucles sur chaque référence de sommation
   if s1 <> exs[1] then
    begin
     for s2 := 0 to maxs[2] do
      if s2 <> exs[2] then
       begin
        for s3 := 0 to maxs[3] do
         if s3 <> exs[3] then
```

```
        begin
         if refvar <> 0
          then for i := 1 to max_indices do
                ind_var[i] := ind_boucle(tab_var[refvar].no_ent_par[i]);
                (* valeur du ième indice de la refvarième variable *)
         if refcst <> 0
           then begin
                 for i := 1 to max_indices do
                   ind_cst[i] := ind_boucle(tab_cst[refcst].no_ent_par[i
                   (* valeur du ième indice de la refcstième constante *
                end;
         if refvar <> 0  (* recherche des particularités éventuelles *)
          then case code of  (* au niveau de la var et de la cst     *)
               1 : begin
                    cherche_cas_special(tab_var[refvar].no_ent_par,
                                        tab_equ[no_equ].code1_var[cpt_ele]
                                        tab_equ[no_equ].val1_var[cpt_ele],
                                        ind_var);
                    if refcst <> 0 then
                     cherche_cas_special(tab_cst[refcst].no_ent_par,
                                         tab_equ[no_equ].code1_cst[cpt_ele
                                         tab_equ[no_equ].val1_cst[cpt_ele]
                                         ind_cst);
                   end;
               2 : if refcst <> 0 then
                     cherche_cas_special(tab_cst[refcst].no_ent_par,
                                         tab_equ[no_equ].code2_cst[cpt_ele
                                         tab_equ[no_equ].val2_cst[cpt_ele]
                                         ind_cst);
               3 : begin
                    cherche_cas_special(tab_var[refvar].no_ent_par,
                                        fo.code_var[cpt_ele],
                                        fo.val_var[cpt_ele],
                                        ind_var);
                    if refcst <> 0
                     then cherche_cas_special(tab_cst[refcst].no_ent_par
                                              fo.code_cst[cpt_ele],
                                              fo.val_cst[cpt_ele],
                                              ind_cst);
                   end;
              end;
         if refcst <> 0
          then valeur := mult *
                         val_cst(refcst,ind_cst[1],ind_cst[2],ind_cst[3])
          else valeur := mult;
         case code of
          1 : pour_a;
          2 : pour_b;
          3 : pour_c;
         end;
        end;
       end;
     end;
 y('boucles_som');
end;
```

```
(*************************************************************************)
(* procédure initialisant et effectuant les boucles sur les familles *)
(* de sommation pour la matrice des premiers membres                     *)
(*************************************************************************)

procedure a_init(no_equ : integer);
 begin
  x('a init');
  for cpt_ele := 1 to nb1_som[no_equ] do
   begin
    pour_som(tab_equ[no_equ].no1_ent_som[cpt_ele],
             tab_equ[no_equ].code1_som[cpt_ele],
             tab_equ[no_equ].val1_som[cpt_ele]);
    mult := tab_equ[no_equ].mult1[cpt_ele];
    refcst := tab_equ[no_equ].cst1[cpt_ele];
    refvar := tab_equ[no_equ].var1[cpt_ele];
    boucles_som;
   end;
   y('a init');
 end;

(*************************************************************************)
(* procédure initialisant et effectuant les boucles sur les familles *)
(* de sommation pour le vecteur des seconds membres des contraintes  *)
(*************************************************************************)

procedure b_init(no_equ : integer);
 begin
  x('b init');
  for cpt_ele := 1 to nb2_som[no_equ] do
   begin
    pour_som(tab_equ[no_equ].no2_ent_som[cpt_ele],
             tab_equ[no_equ].code2_som[cpt_ele],
             tab_equ[no_equ].val2_som[cpt_ele]);
    mult := tab_equ[no_equ].mult2[cpt_ele];
    refcst := tab_equ[no_equ].cst2[cpt_ele];
    boucles_som;
   end;
   y('b init');
 end;

(*************************************************************************)
(* procédure initialisant et effectuant les boucles sur les familles *)
(* de sommation pour le vecteur des coûts                                *)
(*************************************************************************)

procedure c_init;
 begin
  x('c init');
  for cpt_ele := 1 to nb_fo_som do
   begin
    pour_som(fo.no_ent_som[cpt_ele],
             fo.code_som[cpt_ele],
             fo.val_som[cpt_ele]);
    mult := fo.mult[cpt_ele];
    refcst := fo.cst[cpt_ele];
    refvar := fo.variable[cpt_ele];
```

```
    boucles_som;
   end;
   y('c init');
  end;

  begin (* boucles_ptt *)
   x('boucles_ptt');
   for p1 := 0 to maxp[1] do  (* boucle sur chaque référence de quantif *)
    if p1 <> exp[1] then
    begin
     for p2 := 0 to maxp[2] do
      if p2 <> exp[2] then
      begin
       for p3 := 0 to maxp[3] do
        if p3 <> exp[3] then
        begin
         case code of
         1 : a_init(no_equ);
         2 : b_init(no_equ);
         3 : c_init;
         end;
          if (code <> 3)
           then begin
                 no_ligne := no_ligne + 1;
                end;
       end;
      end;
    end;
    y('boucles_ptt');
  end;

 (*************************************************************************)
 (* initialisation des boucles relatives à la quantification de la  *)
 (* contrainte NO_EQU en tenant compte de leurs particularités      *)
 (*************************************************************************)

 procedure pour_tout(no_equ : integer);
  var cpt : integer;
  begin
   x('pour tout');
   for cpt := 1 to max_indices do
    begin
     ptt[cpt] := tab_equ[no_equ].no_ent_ptt[cpt];
     exp[cpt] := - 1;
     if ptt[cpt] = 0
      then maxp[cpt] := 0
      else begin
            maxp[cpt] := largeur(ptt[cpt]) - 1;
            case tab_equ[no_equ].code_ptt[cpt] of
            0,3 : ; (* pas de trou *)
            1,2 : exp[cpt] := tab_equ[no_equ].val_ptt[cpt] -
                              tab_ind[ptt[cpt]].borne_inf;
                               (* no de l'énumération  *)
            end;
           end;
    end;
    y('pour tout');
  end;
```

```
(************************************************)
(* initialisation de l'ensemble des variables *)
(* intervenant dans le simplexe               *)
(************************************************)

procedure init_abc;
 var i,j : integer;
 begin
  x('init_abc');
  for i := 1 to max_li do
   begin
    b[i] := 0.0;
    for j := 1 to max_col do
     a[i,j] := 0.0;
   end;
  for i := 1 to max_col do
   begin
    c[i] := 0.0;
    d[i] := 0.0;
   end;
  z := 0.0;
  w := 0.0;
  y('init_abc');
 end;

begin (* compl_init *)
 x('compl init');
 init_abc;
 (* remplissage de la matrice des premiers membres des contraintes *)
 no_ligne := 1;
 for no := 1 to nb_equ do
  begin
   pour_tout(no);
   boucles_ptt(1,no);
  end;
 (* remplissage du vecteur des seconds membres *)
 no_ligne := 1;
 for no := 1 to nb_equ do
  begin
   pour_tout(no);
   boucles_ptt(2,no);
  end;
 (* remplissage du vecteur des coûts *)
 maxp[1] := 0; maxp[2] := 0; maxp[3] := 0;
 exp[1] := -1; exp[2] := -1; exp[3] := -1;
 boucles_ptt(3,no);
 (* affiche_result_4; *)
 y('compl init');
end;

(************************************************************)
(* procédure d'interfaçage fournissant aux procédures de *)
(* transformation de structures une fonction objectif et *)
(* des contraintes dont tous les éléments sont présentés *)
(* par ordre croissant de références aux variables       *)
(************************************************************)
```

```
procedure interface;
 var cnt,cpt : integer;
 begin
  x('interface');
  tri_fo;
  for cnt := 1 to nb_equ do
   begin
    tri(nb1_som[cnt],cnt);
   end;
   (*affiche_result_3;*)
   y('interface');
 end;

(*********************************************************)
(* procédure inversant le signe des multiplicateurs      *)
(* de la contrainte no NO_EQU                            *)
(*********************************************************)

procedure inverse_signe(no_equ : integer);
 var cnt : integer;
 begin
  x('inverse signe');
  for cnt := 1 to nb1_som[no_equ] do
   tab_equ[no_equ].mult1[cnt] := - tab_equ[no_equ].mult1[cnt];
  for cnt := 1 to nb2_som[no_equ] do
   tab_equ[no_equ].mult2[cnt] := - tab_equ[no_equ].mult2[cnt];
  y('inverse signe');
 end;

(*****************************************************************)
(* procédure permettant de multiplier la contrainte no NO-EQU    *)
(* par -1, en ce compris le retournement du signe d'inégalité    *)
(* si nécessaire; si NO_EQU = 0, on multiplie l'ensemble des     *)
(* contraintes (>=) par -1                                       *)
(*****************************************************************)

procedure mult_par_moins_un(no_equ : integer);
 var cnt,cpt : integer;
 begin
 x('mult par moins un');
  if no_equ <> 0
   then begin
         case tab_equ[no_equ].code_equ of
          2 : tab_equ[no_equ].code_equ := 3;
          3 : tab_equ[no_equ].code_equ := 2;
         end;
         inverse_signe(no_equ);
        end
   else for cpt := 1 to nb_equ do
         begin
          tab_equ[cpt].code_equ := 3; (* au lieu de 2 *)
          inverse_signe(cpt);
         end;
  y('mult par moins un');
 end;
```

```
(*****************************************************************)
(* calcul du nombre de ligne occupé par chaque contrainte en  *)
(* tenant compte de l'éventuelle démultiplication due aux     *)
(* possibilités de la quantification universelle              *)
(*****************************************************************)

procedure calcul_equ;
 var cpt,cnt,nb : integer;
 begin
  x('calcul equ');
  for cpt := 1 to nb_equ do
   begin
    nb := 1;
    for cnt := 1 to max_indices do
     if tab_equ[cpt].no_ent_ptt[cnt] <> 0
      then if tab_equ[cpt].code_ptt[cnt] <> 0
             then nb := nb * (largeur(tab_equ[cpt].no_ent_ptt[cnt]) - 1)
             else nb := nb * largeur(tab_equ[cpt].no_ent_ptt[cnt]);
    place_equ[cpt] := nb;
   end;
   y('calcul equ ');
  end;

(*****************************************************************)
(* procédure d'attribution automatique des variables d'écart  *)
(* et des variables artificielles en fonctions de la nature   *)
(* la contrainte                                              *)
(*****************************************************************)

procedure attrib_va_ve;
 var cnt : integer;
 begin
  x('attrib va ve');
  nb_ve_plus := 0;
  nb_ve_moins := 0;
  nb_va := 0;
  for cnt := 1 to nb_equ do
   begin
    va[cnt] := 0;
    ve[cnt] := 0;
    case tab_equ[cnt].code_equ of
     1 : begin
           va[cnt] := cnt;
           nb_va := nb_va + place_equ[cnt];
         end;
     2 : begin
           ve[cnt] := - cnt;
           nb_ve_moins := nb_ve_moins + place_equ[cnt];
           va[cnt] := cnt;
           nb_va := nb_va + place_equ[cnt];
         end;
     3 : begin
           ve[cnt] := cnt;
           nb_ve_plus := nb_ve_plus + place_equ[cnt];
         end;
    end;
   end;
   y('attrib va ve');
 end;
```

```
(*****************************************************************)
(* procédure permettant l'attribution des variables d'écart et *)
(* artificielles et leur placement correct dans le tableau du  *)
(* simplexe                                                      *)
(*****************************************************************)

procedure aide_init;
 var col_prise,li_prise,cpt,cnt,i,j : integer;

 (*****************************************************************)
 (* procédure d'affectation de l'entier NB dans le tableau du  *)
 (* simplexe et d'affectation de la valeur 1 dans le tableau   *)
 (* des coûts du problème auxiliaire, selon la valeur de CODE  *)
 (* génération du message d'erreur no 72 en cas de dépassement *)
 (* des limites du tableau du simplexe                         *)
 (*****************************************************************)

 procedure affectation(nb,code : integer);
  begin
  x('affectation');
   i := li_prise + 1;
   j := col_prise + 1;
   repeat
    if (i > max_li) or (j > max_col) then erreur(72);
    a[i,j] := nb;
    if code <> 0 then d[j] := 1;
    i := i + 1;
    j := j + 1;
   until i > li_prise + place_equ[cpt];
   col_prise := col_prise + place_equ[cpt];
   y('affectation');
  end;

 (*****************************************************************)
 (* procédure plaçant les variables d'écart dans le tableau du *)
 (* simplexe                                                      *)
 (*****************************************************************)

 procedure ve_init;
  begin
   x('ve init');
   for cpt := 1 to nb_equ do
    begin
     li_prise := 0;
     for cnt := 1 to cpt - 1 do li_prise := li_prise + place_equ[cnt];
     if ve[cpt] > 0 then affectation(1,0)
                    else if ve[cpt] < 0 then affectation(-1,0);
    end;
    y('ve init');
  end;

 (*****************************************************************)
 (* procédure plaçant les variables artificielles dans le      *)
 (* tableau du simplexe                                           *)
 (*****************************************************************)
```

```
 procedure va_init;
  begin
   x('va init');
   for cpt := 1 to nb_equ do
    begin
     li_prise := 0;
     for cnt := 1 to cpt - 1 do li_prise := li_prise + place_equ[cnt];
     if va[cpt] <> 0 then affectation(1,1);
    end;
    y('va init');
  end;

 begin (* aide_init *)
  x('aide init');
  col_prise := 0;
  for cpt := 1 to nb_var do col_prise := col_prise + place_var[cpt];
  ve_init;
  va_init;
  y('aide init');
 end;

(*****************************************************************)
(* procédure initialisant complètement et correctement le  *)
(* problème auxiliaire                                           *)
(*****************************************************************)

procedure maj_va;
 var cpt,cnt,li_prise : integer;
 begin
 x('maj va');
  for cpt := 1 to nb_equ do
   begin
    li_prise := 0;
    for cnt := 1 to cpt - 1 do li_prise := li_prise + place_equ[cnt];
    if va[cpt] <> 0
     then for i := li_prise + 1 to li_prise + place_equ[cpt] do
           begin
            for j := 1 to nb_col do d[j] := d[j] - a[i,j];
            w := w - b[i];
           end;
   end;
   y('maj va');
 end;

(*****************************************************************)
(* procédure d'initialisation complète du tableau du simplexe *)
(* et des vecteurs associés                                      *)
(*****************************************************************)

procedure init_simpl;
 var i,j : integer;
 begin
  x('init simpl');
  compl_init;
  (* affiche_result_4; *)
  aide_init;
  (* affiche_result_4; *)
  maj_va;
```

```
  for j := 1 to nb_col do c_bis[j] := c[j];
  y('init simpl');
 end;

(*****************************************************************)
(* procédure permettant de mémoriser les numéros des colonnes *)
(* qui à chaque étape contiennent l'inverse de la base        *)
(*****************************************************************)

procedure constit_inv_base;
 var cpt,cnt : integer;
 begin
  x('constit inv base');
  for cpt := 1 to max_li do inverse_base[cpt] := 0;
  for cpt := nb_col - nb_ve_plus- nb_ve_moins- nb_va + 1 to nb_col do
   if col_base(cpt) then inverse_base[mem_un[cpt]] := cpt;
  y('constit inv base');
 end;

(*************************************************)
(* fonction booléenne renvoyant VRAI si toutes les *)
(* contraintes sont de type >=                     *)
(*************************************************)

function cond_dual : boolean;
 var cnt : integer;
 begin
 x('cond dual');
  cond_dual := true;
  for cnt := 1 to nb_equ do
   begin
    if tab_equ[cnt].code_equ <> 2
     then cond_dual := false;
   end;
  y('cond dual');
 end;

(************************************************)
(* procédure effectuant les changements de signe *)
(* nécessaires dans la fonction objectif lorsque *)
(* le problème à traiter est une maximisation    *)
(************************************************)

procedure max_vers_min;
 var cnt : integer;
 begin
 x('max vers min');
  if not fo.nature
   then begin
         chge_fo := true;
         for cnt := 1 to nb_fo_som do fo.mult[cnt] := - fo.mult[cnt];
        end;
  y('max vers min');
 end;
```

```
(*************************************************************)
(* procédure renvoyant dans CHERCHE le numéro de la colonne *)
(* qui, parmi les MAX-CNT premières colonnes de COL_CHERCHE,*)
(* contient le réel le plus négatif                          *)
(* s'il n'y a pas de réel négatif, CHERCHE est nul           *)
(*************************************************************)

procedure col_rech_prem(max_cnt : integer;
                        col_teste : reel_col;
                        var cherche : integer);
 var cnt : integer;
     chg : boolean;

 (*****************************************)
 (* procédure de recherche proprement dite *)
 (*****************************************)

 procedure proc;
  begin
   x('proc col');
   if col_teste[cnt] < 0
         then begin
               if col_teste[cnt] < col_teste[cherche]
               then begin
                     cherche := cnt;
                     chg := true;
                    end;
              end;
   y('proc col');
  end;

 begin (* col_rech_prem *)
  x('col rech prem');
  cnt := 1;
  cherche := 1;
  chg := false;
  repeat
   proc;
   cnt := cnt + 1
  until cnt > max_cnt;
  if (not chg) and (col_teste[1] >= 0)
   then cherche := 0;
  y('col rech prem');
 end;

(*************************************************************)
(* procédure renvoyant dans CHERCHE le numéro de la ligne qui,*)
(* parmi les MAX-CNT premières lignes non redondantes de LI_TESTE *)
(* contient le réel le plus négatif                          *)
(* s'il n'y a pas de réel négatif, CHERCHE est nul           *)
(*************************************************************)

procedure li_rech_prem(max_cnt : integer;
                       li_teste : reel_li;
                       var cherche : integer);
 var cnt : integer;
     chg : boolean;
```

```
(*****************************************)
(* procédure de recherche proprement dite *)
(*****************************************)

procedure proc;
 begin
  x('proc li');
  if li_teste[cnt] < 0
        then if li_teste[cnt] < li_teste[cherche]
              then begin
                    cherche := cnt;
                    chg := true;
                   end;
  y('proc li');
 end;

begin (* li_rech_prem *)
 x('li rech prem');
 cnt := 1;
 cherche := 1;
 chg := false;
 repeat
  if not li_redond[cnt] then proc;
  cnt := cnt + 1
 until cnt > max_cnt;
 if (not chg) and (li_teste[1] >= 0)
  then cherche := 0
  else begin
        if not chg
         then begin
               if not li_redond[1]
                then cherche := 1
                else cherche := 0;
              end;
       end;
 y('li rech prem');
end;

(*************************************************************)
(* fonction booléenne renvoyant VRAI si tous les coûts sont >= 0  *)
(*************************************************************)

function hyp_dual : boolean;
 var cnt : integer;
 begin
  x('hyp dual');
  hyp_dual := true;
  for cnt := 1 to nb_col do
   if c[cnt] < 0 then hyp_dual := false;
  y('hyp dual');
 end;

(*************************************************************)
(* procédure rendant tous les seconds membres des contraintes >= 0 *)
(*************************************************************)
```

```
procedure hyp_primal;
 var cnt : integer;
 begin
  x('hyp primal');
  for cnt := 1 to nb_li do
   if b[cnt] < 0
    then mult_par_moins_un(cnt);
  y('hyp primal');
 end;

(*****************************************************************)
(* fonction booléenne renvoyant la valeur VRAI si toutes les *)
(* contraintes sont de type <=                                   *)
(*****************************************************************)

function hyp_une_phase : boolean;
 var cnt : integer;
 begin
  x('hyp une phase');
  hyp_une_phase := true;
  for cnt := 1 to nb_equ do
   if tab_equ[cnt].code_equ <> 3
    then hyp_une_phase := false;
  y('hyp une phase');
 end;

(*************************************************************)
(* procédure mettant en oeuvre l'algorithme du simplexe *)
(*************************************************************)

procedure une_phase(nb : integer);
 begin
  x('une phase');
  repeat
   col_rech_prem(nb,c,col_piv);
   if col_piv <> 0
    then begin
          prim_rech_sort;
          pivotage(l_piv,col_piv,1);
         end;
  until col_piv = 0;
  y('une phase');
 end;

(*************************************************)
(* procédure agençant l'étude complète d'une *)
(* éventuelle ligne redondante                   *)
(*************************************************)

procedure etude_de_cas;
 var rsq_redond : boolean;

 (*****************************************************************)
 (* procédure de detection des lignes risquant d'être redondantes *)
 (* croisement entre la colonne d'une variable entrante et les    *)
 (* lignes éventuelles des variables artificielles en base        *)
 (*****************************************************************)
```

```
procedure test_redond;
 var cpt : integer;
 begin
 x('tst redond');
  rsq_redond := true;
  cpt := nb_col - nb_va + 1;
  repeat
   if col_base(cpt)
    then if abs(a[mem_un[cpt],col_piv]) >= epsilon
          then begin
                rsq_redond := false;
               end;
   cpt := cpt + 1;
  until (not rsq_redond) or (cpt > nb_col);
  y('tst redond');
 end;

(*************************************************************************)
(* détection des lignes redondantes par analyse du croisement des *)
(* colonnes des variables non artificielles  et des lignes des    *)
(* variables artificielles en base                                *)
(*************************************************************************)

procedure trt_redond;
 var cpt,cnt : integer;
 begin
  x('trt redond');
  cpt := nb_col - nb_va + 1;
  repeat
   if col_base(cpt)
    then begin
          cnt := 1;
          repeat
           if abs(a[mem_un[cpt],cnt]) >= epsilon
            then rsq_redond := false;
           cnt := cnt + 1;
          until (not rsq_redond) or (cnt > nb_col - nb_va);
          if rsq_redond then begin
                              li_redond[mem_un[cpt]] := true;
                             end;

         end;
   cpt := cpt + 1;
  until cpt > nb_col;
  y('trt redond');
 end;

begin (* etude_de_cas *)
 x('etude de cas');
 test_redond;
 if rsq_redond
  then trt_redond;
  y('etude de cas');
end;
```

```
(*****************************************************************)
(* dans le cadre de la méthode des deux phases,                  *)
(* procédure permettant de sortir une variable artificielle      *)
(* en base en faisant entrer une n'importe quelle variable       *)
(* non artificielle                                              *)
(* production du message no 74 si c'est impossible               *)
(*****************************************************************)

procedure piv_va(no_li : integer);
 var cpt,no_col : integer;
 begin
  x('piv va');
  cpt := 1;
  no_col := 0;
  repeat
   if not col_base(cpt) then begin
                              no_col := cpt;
                              pivotage(no_li,no_col,1);
                             end;
   cpt := cpt + 1;
  until (no_col <> 0) or (cpt > nb_col - nb_va + 1);
 if cpt > nb_col - nb_va + 1 then erreur(74);
 y('piv va');
end;

(*****************************************************************)
(* dans le cadre de la méthode des deux phases,                  *)
(* procédure de détection des variables artificielles en base    *)
(* et s'il y en a, étude des implications de leur présence       *)
(*****************************************************************)

procedure sortir_va;
 var cpt : integer;
     pas_va : boolean;
 begin
  x('sortir va');
  cpt := nb_col - nb_va + 1;
  repeat
   pas_va := true;
   repeat
    if col_base(cpt) then pas_va := false;
    cpt := cpt + 1;
   until (not pas_va) or (cpt > nb_col);
   if pas_va then une_phase(nb_col - nb_va)
             else begin
                   repeat
                    col_rech_prem(nb_col - nb_va,c,col_piv);
                    if col_piv <> 0
                     then begin
                           etude_de_cas;
                           if not li_redond[cpt - 1 - nb_col + nb_va]
                            then piv_va(cpt - 1 - nb_col + nb_va)
                            else une_phase(nb_col - nb_va);
                          end;
                   until col_piv = 0;
                  end;
  until pas_va;
  y('sortir va')
 end;
```

```
(*************************************************************)
(* procédure mettant en oeuvre la méthode du simplexe,       *)
(* la méthode des deux phases et ses apports en matière      *)
(* de détection de redondances et de non compatibilité       *)
(* production du message no 75 dans ce dernier cas           *)
(*************************************************************)

procedure primal;
 begin
  x('primal');
  if not hyp_une_phase
   then begin
         repeat
          col_rech_prem(nb_col,d,col_piv);
          if col_piv = 0
           then if w <> 0
                 then erreur(75)
                 else begin
                       sortir_va;
                      end
           else begin
                 prim_rech_sort;
                 pivotage(l_piv,col_piv,2);
                end;
         until col_piv = 0;
        end
   else begin
         une_phase(nb_col); (* une seule phase *)
        end;
  y('primal');
 end;

(*********************************************************************)
(* procédure mettant en oeuvre l'algorithme dual du simplexe         *)
(*********************************************************************)

procedure dual;
 begin
  x('dual');
  repeat
   li_rech_prem(nb_li,b,l_piv);
   if l_piv <> 0
    then begin
          dual_rech_entr;
          pivotage(l_piv,col_piv,1);
         end;
  until l_piv = 0;
  y('dual');
 end;

(*********************************************************************)
(* procédure prenant en charge le compte-rendu des résultats         *)
(* signalisation des lignes redondantes, fonction objectif,          *)
(* variables en base et leurs valeurs                                *)
(*********************************************************************)
```

```
procedure resultat;
 var cpt,nb,cnt : integer;
 begin
  x('resultat');
  genere_nom_equ;
  genere_nom_var;
  ve_nom;
  va_nom;
  nb := 0;
  clrscr;
  for cpt := 1 to nb_li do
   begin
    if li_redond[cpt]
     then ecrisln('  La contrainte n° ' + i2s(cpt,3) + ' est redondante.');
   end;
  cpt := 1;
  ecrisln('');
  ecrisln('  Solution optimale : ');
  if not ecran
   then msg('Solution optimale',5,false);
  if not chge_fo then z := -z;
  ecrisln('');
  ecrisln('          Fonction objectif : z = ' + r2s(z,10,5));
  ecrisln('');
  cnt := 1;
  window(2,15,78,23);
  gotoxy(1,1);
  repeat
   if col_base(cpt)
    then begin
          nb := nb + 1;
          ecrisln(s2s(var_nom[cpt],48) + ' = ' + r2s(b[mem_un[cpt]],10,5));
          if ecran then cnt := cnt + 1;
          if cnt = 7 then begin
                          gotoxy(30,8);
                          write('poussez sur <RETURN> pour la suite : ');
                          readln;
                          cnt := 1;
                          clrscr;
                         end;
         end;
   cpt := cpt + 1;
  until nb >= nb_li;
  gotoxy(30,8);
  write('poussez sur <RETURN> pour la suite : ');
  readln;
  window(2,14,78,23);
  clrscr;
  gotoxy(1,1);
  ecrisln(' Résultats sur les coûts : ');
  window(2,15,78,23);
  gotoxy(1,1);
  cpt := 1;
  cnt := 1;
  repeat
   if not col_base(cpt)
    then begin
          ecrisln(s2s(var_nom[cpt],48) + ' : ' + r2s(c[cpt],10,5));
          if ecran then cnt := cnt + 1;
         end;
```

```
   if cnt = 7
    then begin
          gotoxy(30,8);
          write('poussez sur <RETURN> pour la suite : ');
          readln;
          cnt := 1;
          clrscr;
         end;
    cpt := cpt + 1;
   until cpt > nb_col;
   if ecran
    then begin
          gotoxy(30,8);
          write('poussez sur <RETURN> pour la suite : ');
          readln;
         end;
   window(2,5,78,23);
   gotoxy(1,1);
   y('resultat');
  end;

begin (* calcul_et_simplexe *)
 x('calcul et simplexe');
 msg('calcul du simplexe en cours',8,false);
 interface;
 calcul_equ;
 max_vers_min;
 det_place_col;
 det_place_li;
 for cpt := 1 to nb_li do li_redond[cpt] := false;
 compl_init;
 if hyp_dual (* tous couts positifs *)
  then begin
        if cond_dual then begin (* toutes equations >= *)
                           mult_par_moins_un(0);
                           attrib_va_ve;
                           nb_col := nb_col + nb_ve_plus +
                                     nb_ve_moins + nb_va;
                           if nb_col > max_col then erreur(72);
                           init_simpl;
                           constit_inv_base;
                           dual;
                          end
                     else begin
                           hyp_primal;
                           attrib_va_ve;
                           nb_col := nb_col + nb_ve_plus +
                                     nb_ve_moins + nb_va;
                           if nb_col > max_col then erreur(72);
                           init_simpl;
                           constit_inv_base;
                           primal;
                          end;
       end
  else begin
        hyp_primal; (* rendre tous b[i] positifs *)
        attrib_va_ve;
        nb_col := nb_col + nb_ve_plus +
                  nb_ve_moins + nb_va;
```

```
        if nb_col > max_col then erreur(72);
        init_simpl;
        constit_inv_base;
        primal;
       end;
  resultat; (* a n'ecrire que si on a obtenu une solution optimale *)
  y('calcul et simplexe');
end;

(*****************************************************************)
(* procédure permettant le choix d une analyse de sensibilité ou *)
(* d'un paramétrage, les éléments sur lesquels effectuer ce      *)
(* traitement et le traitement proprement dit avec compte-rendu  *)
(* résultats                                                     *)
(*****************************************************************)

overlay procedure sensibilite_et_parametrage; (* overlay de niveau 2 *)

(*************************************************************)
(* procédure renvoyant le message no 71 en cas de création *)
(* dynamique de records dépassant la capacité mémoire      *)
(*************************************************************)

procedure mem_disponible;
 begin
  if memavail < 28 then erreur(71);
 end;

(*************************************************************)
(* procédure permettant de retrouver à partir de la        *)
(* mémorisation de l'ensemble des coûts initiaux, les coûts*)
(* initiaux des variables en base à une étape donnée du    *)
(* développement dans l'ordre où les variables sont en base*)
(*************************************************************)

procedure cout_base;
 var cpt : integer;
 begin
  x('cout base');
  for cpt := 1 to nb_col do
   if col_base(cpt)
    then base[mem_un[cpt]] := cpt;
  for cpt := 1 to nb_li do c_base[cpt] := c_bis[base[cpt]];
  y('cout base');
 end;

(*****************************************************************)
(* procédure déterminant l'intervalle dans lequel la solution   *)
(* obtenue reste valable; LIM_INF et LIM_SUP en sont les bornes *)
(* DANGER_INF et DANGER_SUP sont les n° des colonnes qui        *)
(* provoqueraient un changement de base, VALEUR est la valeur   *)
(* actuelle testée pour la détermination de l'intervalle        *)
(* CNT est la colonne correspondante, CODE et PARAM ne sont     *)
(* utilisés que pour le paramétrage, respectivement pour        *)
(* distinguer paramétrage sur les coûts et sur les seconds      *)
(* membres et pour indiquer la colonne ou la ligne analysée     *)
(*****************************************************************)
```

```
procedure intervalle(code,param,cnt : integer;
                     valeur : real;
                     var danger_inf,danger_sup : integer;
                     var lim_inf,lim_sup : real);
 begin
  x('intervalle');
  (*
  case code of
   0 : begin
        lim_inf := c_lim_inf[param];
        lim_sup := c_lim_sup[param];
       end;
   1 : begin
        lim_inf := b_lim_inf[param];
        lim_sup := b_lim_sup[param];
       end;
  end;
  *)
  if valeur > 0
   then begin
         if valeur < lim_sup
          then begin
                lim_sup := valeur;
                danger_sup := cnt;
               end;
        end
   else begin
         if valeur < 0
          then begin
                if valeur > lim_inf
                 then begin
                       lim_inf := valeur;
                        danger_inf := cnt;
                      end;
               end;
        end;
  y('intervalle');
 end;

(*****************************************************************)
(* calcul de la pente de la fonction objectif (partie      *)
(* dépendant du paramètre) en cas de traitement du second *)
(* membre d'une contrainte                                        *)
(*****************************************************************)

procedure b_pente;
 var cpt : integer;
 begin
  x('b_pente');
  pente := 0;
  for cpt := 1 to nb_li do
   begin
    pente := pente + c_base[cpt] * b_delta[cpt];
   end;
  y('b pente');
 end;
```

```
(*************************************************************************)
(* écriture de la solution de l'analyse de sensibilité sur la ligne *)
(* NO_PARAM dans l'intervalle [INF,SUP]                                  *)
(*************************************************************************)

procedure b_sol_sensib(no_param : integer;
                       inf,sup : real);
 begin
  clrscr;
  ecrisln('');
  ecrisln(' Analyse de sensibilite sur la ligne ' + i2s(no_param,2) + ' : ')
  ecrisln('');
  ecrisln(' La solution reste valable pour le paramètre DELTA dans');
  if inf = - maxreal
   then ecris('   [-∞')
   else ecris('   [' + r2s(inf,10,5));
  ecris(',');
  if sup = maxreal
   then ecrisln('+∞]')
   else ecrisln(r2s(sup,10,5) + ']');
  ecrisln('');
  ecrisln('   avec une pente ' + r2s(pente,10,5));
  attente;
 end;

(*************************************************************)
(* calcul de la pente de la fonction objectif (partie     *)
(* dépendant du paramètre) en cas de traitement d'un coût *)
(* de la fonction objectif                                *)
(*************************************************************)

procedure c_pente(no_param : integer);
 begin
  x('c pente');
  if not col_base(no_param)
   then pente := 0
   else pente := b[mem_un[no_param]];
  y('c pente');
  end;

(*************************************************************************)
(* écriture de la solution de l'analyse de sensibilité sur la colonne *)
(* NO_PARAM dans l'intervalle [INF,SUP]                                  *)
(*************************************************************************)

procedure c_sol_sensib(no_param : integer;
                       inf,sup : real);
 begin
  clrscr;
  ecrisln('');
  ecrisln(' Analyse de sensibilite sur la colonne ' + i2s(no_param,2) + ' :
  ecrisln('');
  ecrisln(' La solution reste valable pour le paramètre DELTA dans');
  if inf = - maxreal
   then ecris('   [-∞')
   else ecris('   [' + r2s(inf,10,5));
  ecris(',');
```

```
  if sup = maxreal
   then ecrisln('+∞]')
   else ecrisln(r2s(sup,10,5) + ']');
  ecrisln('');
  ecrisln('  avec une pente ' + r2s(pente,10,5));
  attente;
 end;

(*************************************************************************)
(* analyse de sensibilité sur le second membre de la contrainte  *)
(* occupant la ligne no NO_PARAM dans le tableau des contraintes *)
(*************************************************************************)

procedure b_sensib(no_param : integer);
 var d_inf,d_sup,no_col,li : integer;
     inf,sup : real;
 begin
  x('b sensib');
  inf := - maxreal;
  sup := maxreal;
  no_col := inverse_base[no_param];
  for i := 1 to nb_li do
   begin
    b_delta[i] := a[i,no_col];
    if b[i] = 0
     then begin
           if a[i,no_col] < 0
            then begin
                  if sup > 0
                   then begin
                         sup := 0;
                         d_sup := i;
                        end;
                 end
            else begin
                  if a[i,no_col] > 0
                   then begin
                         if inf < 0
                          then begin
                                inf := 0;
                                d_inf := i;
                               end;
                        end;
                 end;
          end
     else if a[i,no_col] <> 0
           then begin
                 intervalle(2,0,i,-b[i]/a[i,no_col],d_inf,d_sup,inf,sup);
                end;
   end;
  b_pente;
  b_sol_sensib(no_param,inf,sup);
  y('b sensib');
 end;
```

```
(*************************************************************************)
(* analyse de sensibilité sur le coût de la variable de la fonction *)
(* objectif occupant la colonne NO_PARAM dans le tableau des        *)
(* contraintes et dans le vecteur associé                           *)
(*************************************************************************)

procedure c_sensib(no_param : integer);
 var d_inf,d_sup,no_ligne,j : integer;
     inf,sup : real;
 begin
  x('c sensib');
  inf := - maxreal;
  sup := maxreal;
  if col_base(no_param)
   then begin
         no_ligne := mem_un[no_param];
         for j := 1 to nb_col do
         begin
         if not col_base(j)
          then begin
               if a[no_ligne,j] <> 0
                then intervalle(2,0,j,c[j]/a[no_ligne,j],
                                d_inf,d_sup,inf,sup);
               end;
         end;
        end
   else for j := 1 to nb_col - nb_va do
         if j = no_param
          then if j = 1 then inf := - c[j]
                        else if -c[j] > inf then inf := -c[j];
  c_pente(no_param);
  c_sol_sensib(no_param,inf,sup);
  y('c sensib');
 end;

(*************************************************************************)
(* mise à jour des différentes informations relatives au paramétrage  *)
(* LITIGE_INF et LITIGE_SUP sont les numéros des lignes ou colonnes   *)
(* provoquant le changement de base, VALEUR_INF et VALEUR_SUP sont les*)
(* bornes de l'intervalle de validité de la solution courante, NBRE   *)
(* est la valeur au-delà ou en-deçà de laquelle l'analyse continue,   *)
(* ATTENTION est le numéro de ligne ou de colonne déterminant pour le *)
(* changement de base et CONTROLE indique si on travaille à gauche ou *)
(* à droite de la solution de départ                                  *)
(*************************************************************************)

procedure mise_a_jour(controle,litige_inf,litige_sup : integer;
                      valeur_inf,valeur_sup : real;
                      var attention : integer;
                      var nbre : real);
 begin
  x('mise a jour');
  if controle = 0 then begin
                       nbre := valeur_inf;
                       attention := litige_inf;
                       ptr^.svt := premier;
                       premier := ptr;
                      end
```

```
                    else begin
                          nbre := - valeur_sup;
                          attention := litige_sup;
                          ptr^.svt := dernier;
                          dernier^.svt := ptr;
                          dernier := ptr;
                         end;
  y('mise a jour');
 end;

(*************************************************************************)
(* mémorisation des variables en base et de leurs valeurs dans un *)
(* record caractérisant une solution dans le cadre du paramétrage *)
(*************************************************************************)

procedure val_var_base_fo;
 var cpt,nb : integer;
 begin
 x('val var base fo');
  ptr^.indep := z;
  ptr^.pente := pente;
  cpt := 1;
  nb := 0;
  repeat
   if col_base(cpt)
    then begin
          nb := nb + 1;
          ptr^.col_sol[nb] := cpt;
          ptr^.val_sol[nb] := b[mem_un[cpt]];
         end;
   cpt := cpt + 1;
  until nb >= nb_li;
  y('val var base fo');
 end;

(*******************************************************)
(* écriture des variables en base et de leur valeur *)
(* sur les périphériques choisis                    *)
(* (paramétrage)                                    *)
(*******************************************************)

procedure resbis;
 var cpt : integer;
     strg : str80;
 begin
  x('resbis');
  for cpt := 1 to nb_li do
   begin
    strg := var_nom[ptr^.col_sol[cpt]];
    ecrisln(s2s(strg,48) + ' = ' + r2s(ptr^.val_sol[cpt],10,5));
   end;
   y('resbis');
 end;
```

```
(*****************************************************)
(* écriture de la solution valable dans [NO1,NO2] *)
(* (paramétrage)                                     *)
(*****************************************************)

procedure ecr_interv(no1,no2 : real);
 begin
 x('ecr interv');
  if ptr^.code1 = 2
   then begin
         ecris('Dans l''intervalle [');
         if no1 = - maxreal
          then ecris('-∞')
          else ecris(r2s(no1,10,5));
         ecris(',');
         if no2 = maxreal
          then ecrisln('+∞], ')
          else ecrisln(r2s(no2,10,5) + '], ');
         ecrisln('');
         ecrisln('  Fonction objectif : z = ' + r2s(ptr^.indep,10,5) +
                 ' + DELTA * ' + r2s(ptr^.pente,10,5));
         ecrisln('');
         resbis; (* ecrire var en base et leur valeur *)
        end
   else non_real_bor;
  y('ecr interv');
 end;

(*****************************************************************)
(* écriture de l'ensemble des résultats du paramétrage en partant *)
(* de la limite inférieure vers la limite supérieure              *)
(*****************************************************************)

procedure ecrire_liste;
 begin
  x('ecrire liste');
  if not chge_fo then z := - z;
  ptr := premier;
  ecr_interv(lim_inf,ptr^.sup);
  ptr := ptr^.svt;
  repeat
   ecr_interv(ptr^.inf,ptr^.sup);
   ptr := ptr^.svt;
  until ptr = dernier; (* a verifier *)
  ecr_interv(ptr^.inf,lim_sup);
  y('ecrire liste');
 end;

(*****************************************************************)
(* LIM étant une des limites fixées par l'utilisateur, la procédure *)
(* effectue les changements de base nécessaires tant que NB  n'a pas*)
(* atteint cette limite; DANGER est le no de la ligne qui provoque  *)
(* le changement de base, D_INF et D_SUP, les bornes de l'intervalle*)
(* dans lequel la nouvelle solution est valable, PARAM est le no de *)
(* la ligne analysée et CODE permet de distinguer traitement des    *)
(* et des seconds membres                                           *)
(*****************************************************************)
```

```
procedure b_valid(var nb: real;
                  lim : real;
                  var danger : integer;
                  code,param : integer;
                  var d_inf,d_sup : integer);
 begin
 x('b valid');
  repeat
   if nb > lim
    then begin
          l_piv := danger;
          dual_rech_entr;
          if col_piv <> 0
           then begin
                 pivotage(l_piv,col_piv,3);
                 mem_disponible;
                 new(ptr);
                 for i := 1 to nb_li do
                  if b_delta[i] <> 0
                   then begin
                         intervalle(1,param,i,-b[i]/b_delta[i],d_inf,d_sup,
                                    ptr^.inf,ptr^.sup);
                        end;
                 val_var_base_fo;
                 mise_a_jour(code,d_inf,d_sup,ptr^.inf,ptr^.sup,danger,nb);
                end;
         end;
  until (nb <= lim) or (col_piv = 0);
  if col_piv = 0 then begin
                       mem_disponible;
                       new(ptr);
                       ptr^.code1 := code;
                       ptr^.code2 := 0;
                       ptr^.nb := nb;
                      end;
  y('b valid');
 end;

(*****************************************************************)
(* procédure de recopie de tous les tableaux nécessaires à la    *)
(* mémorisation de la solution de départ et qui vont être        *)
(* modifiés par le paramétrage                                   *)
(*****************************************************************)

procedure copie_tab;
 var i,j : integer;
 begin
  x('copie tab');
  for i := 1 to nb_li do
   begin
    for j := 1 to nb_col do a_copie[i,j] := a[i,j];
    b_copie[i] := b[i];
    b_delta_copie[i] := b_delta[i];
   end;
  for j := 1 to nb_col do
   begin
    c_copie[j] := c[j];
    c_delta_copie[j] := c_delta[j];
   end;
  z_copie := z;
```

```
  pente_copie := pente;
  y('copie tab');
 end;

(*****************************************)
(* paramétrage sur la ligne no NO_PARAM *)
(*****************************************)

procedure b_param(no_param : integer);
 var retenir : real;
     no_col,i,d_inf,d_sup : integer;
 begin
  x('b param');
  premier := nil;
  dernier := nil;
  no_col := inverse_base[no_param];
  mem_disponible;
  new(ptr);
  for i := 1 to nb_li do
   begin
    b_delta[i] := a[i,no_col];
    if b[i] = 0
     then begin
           if a[i,no_col] < 0
            then begin
                  if ptr^.sup > 0
                   then begin
                         ptr^.sup := 0;
                         d_sup := i;
                        end;
                 end
            else begin
                  if a[i,no_col] > 0
                   then begin
                         if ptr^.inf < 0
                          then begin
                                ptr^.inf := 0;
                                d_inf := i;
                               end;
                        end;
                 end;
          end
     else if a[i,no_col] <> 0
           then begin
                 intervalle(1,no_param,i,-b[i]/a[i,no_col],d_inf,d_sup,
                            ptr^.inf,ptr^.sup);
                end;
    end;
   ptr^.indep := z;
   b_pente;
   ptr^.pente := pente;
   dernier := premier;
   ptr^.svt := premier;
   premier := ptr;
   copie_tab;
   retenir := - ptr^.sup;
```

```
  b_lim_sup[no_param] := - b_lim_sup[no_param];
  b_valid(ptr^.inf,b_lim_inf[no_param],d_inf,0,no_param,d_inf,d_sup);
  b_valid(retenir,b_lim_sup[no_param],d_sup,1,no_param,d_inf,d_sup);
  ecrire_liste;
  y('b param');
 end;

(*************************************************************************)
(* LIM étant une des limites fixées par l'utilisateur, la procédure *)
(* effectue les changements de base nécessaires tant que NB  n'a pas*)
(* atteint cette limite; DANGER est le no de la colonne qui provoque*)
(* le changement de base, D_INF et D_SUP, les bornes de l'intervalle*)
(* dans lequel la nouvelle solution est valable, PARAM est le no de *)
(* la ligne analysée et CODE permet de distinguer traitement des    *)
(* et des seconds membres                                           *)
(*************************************************************************)

procedure c_valid(var nb : real;
                  lim : real;
                  var danger : integer;
                  code,param : integer;
                  var d_inf,d_sup : integer);
 var j : integer;
 begin
  x(' c valid');
  repeat
   if nb > lim
    then begin
          col_piv := danger;
          prim_rech_sort;
          if l_piv <> 0
           then begin
                 pivotage(l_piv,col_piv,4);
                 mem_disponible;
                 new(ptr);
                 for j := 1 to nb_col do
                  begin
                   if c_delta[j] <> 0
                    then begin
                          intervalle(0,param,j,c[j]/c_delta[j],d_inf,
                                     d_sup,ptr^.inf,ptr^.sup);
                         end;
                  end;
                 val_var_base_fo;
                 mise_a_jour(code,d_inf,d_sup,ptr^.inf,ptr^.sup,danger,nb);
                end;
         end;
  until (nb <= lim) or (l_piv = 0);
  if l_piv = 0 then begin
                     mem_disponible;
                     new(ptr);
                     ptr^.code1 := code;
                     ptr^.code2 := 1;
                     ptr^.nb := nb;
                    end;
  y('c valid');
 end;
```

```
(*****************************************)
(* paramétrage sur la colonne no NO_PARAM *)
(*****************************************)

procedure c_param(no_param : integer);
 var retenir : real;
     no_ligne,j,d_inf,d_sup : integer;
 begin
  x('c param');
  premier := nil;
  dernier := nil;
  mem_disponible;
  new(ptr);
  if col_base(no_param)
   then begin
         no_ligne := mem_un[no_param];
         for j := 1 to nb_col - nb_va do
          begin
           if not col_base(j)
            then begin
                  for cpt := 1 to nb_col do
                   begin
                    if cpt <> no_param
                     then c_delta[cpt] := - a[no_ligne,cpt];
                   end;
                  if a[no_ligne,j] <> 0
                   then begin
                         ptr^.inf := c_lim_inf[no_param];
                         ptr^.sup := c_lim_sup[no_param];
                         intervalle(0,no_param,j,c[j]/a[no_ligne,j],
                                    d_inf,d_sup,ptr^.inf,ptr^.sup);
                        end;
                 end;
          end;
         end
   else for j := 1 to nb_col - nb_va do
         begin
          if j = no_param
           then begin
                 c_delta[j] := 1;
                 if j = 1 then ptr^.inf := - c[j]
                          else if -c[j] > ptr^.inf then ptr^.inf := -c[j];
                end;
         end;
  ptr^.indep := z;
  if not col_base(no_param) then pente := 0
                            else pente := b[mem_un[no_param]];
  ptr^.pente := pente;
  dernier := premier;
  copie_tab;
  ptr^.svt := premier;
  premier := ptr;
  retenir := - ptr^.sup;
  c_lim_sup[no_param] := - c_lim_sup[no_param];
  c_valid(ptr^.inf,c_lim_inf[no_param],d_inf,0,no_param,d_inf,d_sup);
  c_valid(retenir,c_lim_sup[no_param],d_sup,1,no_param,d_inf,d_sup);
  ecrire_liste;
  y('c param');
 end;
```

```
(*************************************************************)
(* procédure demandant l'analyse de sensibilité ou           *)
(* le paramétrage sur chaque coût choisi par l'utilisateur   *)
(*************************************************************)

procedure calcul_col(code : integer);
 var cpt : integer;
 begin
  for cpt := 1 to nb_col - nb_va do
   begin
    case code of
     1 : if s_anal_var[cpt] then c_sensib(cpt);
     2 : if p_anal_var[cpt] then c_param(cpt);
    end
   end;
 end;

(*************************************************************************)
(* procédure permettant la mémorisation des choix de l'utilisateur *)
(* en matière d'analyse de sensibilité ou de paramétrage, selon la *)
(* valeur de CODE, la présentation des choix lui étant faite en    *)
(* nommant successivement les noms de les variables du problème    *)
(* compte-tenu des valeurs possibles de leurs indices              *)
(*************************************************************************)

procedure choix_col(code : integer);
 var cpt,cnt,col_prise : integer;
     strg,strwrk : str80;
 begin
  x('choix col');
  col_prise := 0;
  gotoxy(5,10);
  clreol;
  case code of
  1 : begin
       write('Analyse sur le coût de :');
      end;
  2 : begin
       write('Paramétrage sur le coût de :');
      end;
  end;
  for cpt := 1 to nb_var do
   begin
    for cnt := 1 to place_var[cpt] do
     begin
      col_prise := col_prise + 1;
      strg := var_nom[col_prise];
      gotoxy(10,16);
      clreol;
      write(strg:48,' o/n : ');
      case code of
      1 : lis_boolean(68,16,s_anal_var[col_prise]);
      2 : lis_boolean(68,16,p_anal_var[col_prise]);
      end;
     end;
   end;
  y('choix col');
 end;
```

```
(*****************************************************************)
(* procédure demandant l'analyse de sensibilité ou               *)
(* le paramétrage sur chaque second membre de contrainte         *)
(* choisi par l'utilisateur                                      *)
(*****************************************************************)

procedure calcul_li(code : integer);
 var cpt,cnt : integer;
 begin
  for cpt := 1 to nb_li do
   begin
    case code of
     1 : if s_anal_li[cpt] then b_sensib(cpt);
     2 : if p_anal_li[cpt] then b_param(cpt);
    end;
   end;
 end;

(*****************************************************************************)
(* procédure permettant la mémorisation des choix de l'utilisateur           *)
(* en matière d'analyse de sensibilité ou de paramétrage, selon la           *)
(* valeur de CODE, la présentation des choix lui étant faite en              *)
(* nommant successivement les noms de toutes les contraintes,                *)
(* compte-tenu de la quantification                                          *)
(*****************************************************************************)

procedure choix_li(code : integer);
 var cpt,cnt,li_prise,li_avt : integer;
     strg : str80;
 begin
  x('choix li');
  li_prise := 0;
  gotoxy(5,10);
  clreol;
  case code of
  1 : begin
       write('Analyse sur le second membre de :');
      end;
  2 : begin
       write('Paramétrage sur le second membre de :');
      end;
  end;
  for cpt := 1 to nb_equ do
    for cnt := 1 to place_equ[cpt] do
    begin
     li_prise := li_prise + 1;
     if not li_redond[li_prise]
      then begin
            strg := equ_nom[li_prise];
            gotoxy(10,16);
            clreol;
            write(strg:48,' (o/n) : ');
            case code of
            1 : lis_boolean(68,16,s_anal_li[li_prise]);
            2 : lis_boolean(68,16,p_anal_li[li_prise]);
            end;
           end;
    end;
  y('choix li');
 end;
```

```
(*****************************************************************)
(* procédure demandant  le traitement de tous les éléments       *)
(* choisis par l'utilisateur                                     *)
(*****************************************************************)

procedure calcul_li_col;
 var cpt : integer;
 begin
  x('calcul li col');
  for cpt := 1 to 2 do
   begin
    calcul_col(cpt);
    calcul_li(cpt);
   end;
  y('calcul li col');
 end;

begin (* sensibilite_et_parametrage *)
 x('sensib et param');
 lim_sup :=   maxreal;
 lim_inf := - maxreal;
 for cpt := 1 to nb_li do
  begin
   s_anal_li[cpt] := false;
   p_anal_li[cpt] := false;
  end;
 for cpt := 1 to nb_col do
  begin
   s_anal_var[cpt] := false;
   p_anal_var[cpt] := false;
  end;
 choix_analyse;
 if c_sens or b_sens or c_par or b_par
  then begin
         if c_sens then choix_col(1);
         if b_sens then choix_li(1);
         if c_par  then choix_col(2);
         if b_par  then choix_li(2);
         cout_base;
         calcul_li_col;
        end;
 y('sensib et param');
end;

procedure fin_pgm;
 begin
  if disque then close(f_out); (* fermeture du fichier de sortie des *)
                               (* résultats s'il y en a un           *)
  window(1,5,80,25);
  clrscr;
  window(1,1,80,25);
  gotoxy(1,5);                 (* fermeture du cadre à l'écran *)
  write('╚');
  for cpt := 1 to 78 do write('═');
  writeln('╝');
  writeln(' Fin du programme PROLICO 1.1');
  writeln;
 end;
```

```
begin (* calcul *)
 x('calcul');
 calcul_et_simplexe;              (* transformation de structures et simplexe *)
 sensibilite_et_parametrage;      (* analyse de sensibilité et paramétrage    *)
 fin_pgm;
 y('calcul');
end;

begin (* program *)
 debug2 := false;                 (* booléen de debugging                     *)
 compilation;                     (* reconnaissance de la syntaxe             *)
 calcul;                          (* le reste, cf ci-dessus                   *)
end.
```

8.2 TESTS

Afin de garantir à l'utilisateur une fiabilité raisonnable, divers tests ont été effectués. L'approche adoptée reflète le découpage logique du programme :

- test sur la reconnaissance de la syntaxe
  - . familles d'indices
  - . constantes
  - . équations
  - . enchaînement des paragraphes
- test sur la transformation des structures
- test sur les calculs
  - . méthode du simplexe
  - . analyse de sensibilité
  - . (paramétrage)

Dans le but d'automatiser au maximum cette phase du travail, nous avons choisi de faire usage d'une possibilité avantageuse offerte par le Turbo Pascal. En effet, lorsque l'exécution d'un programme est demandée, en l'appelant par son nom, le système charge la version exécutable dans le segment de code. Tout caractère suivant éventuellement le nom du programme dont on appelle la version exécutable est ignoré. En fait, ces caractères sont stockés à une adresse prédéfinie dans le segment de code. Le Turbo Pascal offre la possibilité de déclarer une string que l'on "force" à cette adresse, string qui peut ensuite être manipulée et reconnue. Par conséquent, une ligne de commande sera constituée du nom

du programme à exécuter suivi en l'occurence d'un nom de fichier à lire et d'un nom de fichier où enregistrer les résultats. Ces lignes de commandes sont elles-mêmes regroupées au sein d'un fichier, ce qui permet leur exécution sans exiger la présence continuelle du testeur.

Comme indiqué plus haut, les tests ont été conduits de façon à respecter le découpage logique, ce qui se traduit au niveau du programme par l'imposition d'un point d'arrêt (fermeture des fichiers et sortie) immédiatement après la partie à tester. Une telle méthode semble particulièrement bien adaptée ici en raison de la nature même du problème.

Dans la partie traitant de la reconnaissance de la syntaxe, l'accent a été mis non seulement sur l'acceptation de fichiers corrects sous diverses formes mais aussi sur la génération du message d'erreur attendu en fonction de la faute commise et de ses caractéristiques. L'ensemble des résultats est exposé plus bas. Certains des fichiers illustrent des erreurs facilement évitables mais d'autres mettent en évidence des points plus subtils et peuvent être un complémént intéressant au chapitre de définition de la syntaxe dans la mesure où un exemple est parfois plus parlant qu'un ensemble de phrases descriptives.

Les tests sur la transformation des structures ont été conçus de manière à couvrir un nombre représentatif de situations illustrant les diverses possibilités rencontrées et ont été menés, pour des raisons pratiques à la suite des tests relatifs au paragraphe regroupant

fonction objectif et contraintes. On vérifiera l'exactitude de la transformation dans chaque cas à condition que la partie effectivement visible des tableaux dans ce compte-rendu le permette.

Enfin, les divers modules de calcul ont été testés au moyen d'un ensemble de fichiers qui ont pu être aisément vérifiés "à la main". Ajoutons cependant qu'à ce niveau les erreurs d'arrondis doivent être prises en compte. Rappelons enfin que le paramétrage n'a pu faire l'objet de tests suffisants.

On vérifiera aisément que chaque message d'erreur est apparu au moins une fois dans l'ensemble des fichiers de tests.

L'exemple développé dans l'introduction a également été traité dans le cadre de ces tests mais présente un dépassement de taille pour la matrice et les vecteurs utilisés.

TESTS SUR LES FAMILLES

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM000.PAS"

```
fam : I = [1_3];
      J = [-2_2];
      K = [+2_4];
      L = [2_+4];
      M = [+2_+4];
      N = [-5_-3];
      O = [-5_+3];
      ville = (namur, bruxelles, liege);
      province = (namur, brabant, liege);
      impair = (un,trois,cinq,sept);
fin :

paragraphe "familles" traité

10 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de    -2 a     2

  3 k
par intervalle :
de     2 a     4

  4 l
par intervalle :
de     2 a     4

  5 m
par intervalle :
de     2 a     4

  6 n
par intervalle :
de    -5 a    -3

  7 o
par intervalle :
de    -5 a     3

  8 ville
par enumeration :
namur bruxelles liege
```

```
  9 province
par enumeration :
namur brabant liege

 10 impair
par enumeration :
un trois cinq sept
```

```
TRAITEMENT DU FICHIER "B:\FAMILLE\FAM001.PAS"
```

```
I = [1_3];

  *** erreur ( 3) du type nom de paragraphe attendu ***

I = [1_3];
  ^
```

```
TRAITEMENT DU FICHIER "B:\FAMILLE\FAM002.PAS"
```

```
ind : I = [1_3];

  *** erreur ( 3) du type nom de paragraphe attendu ***

ind : I = [1_3];
    ^
```

```
TRAITEMENT DU FICHIER "B:\FAMILLE\FAM003.PAS"
```

```
fam I = [1_3];

  *** erreur ( 3) du type nom de paragraphe attendu ***

fam I = [1_3];
      ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM004.PAS"

```
fam : 123 = [1_3];

        *** erreur ( 1) du type nom invalide ***

fam : 123 = [1_3];
      ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM005.PAS"

```
fam : I [1_3];

         *** erreur (14) du type "=" attendu ***

fam : I [1_3];
      ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM006.PAS"

```
fam : I = 1_3;

     *** erreur (15) du type "(" ou "[" attendu ***

fam : I = 1_3;
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM007.PAS"

```
fam : I = (1_3);

         *** erreur (22) du type "," attendu ***

fam : I = (1_3);
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM008.PAS"

```
fam : I = [1.5_3];
```

*** erreur (8) du type borne inferieure invalide ***

```
fam : I = [1.5_3];
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM009.PAS"

```
fam : I = [1 3];
```

*** erreur (56) du type entier ou "_" attendu ***

```
fam : I = [1 3];
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM010.PAS"

```
fam : I = [1_3.5];
```

*** erreur (7) du type borne superieure invalide ***

```
fam : I = [1_3.5];
            ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM011.PAS"

```
fam : I = [1_3;
```

*** erreur (57) du type entier ou "]" attendu ***

```
fam : I = [1_3;
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM012.PAS"

```
fam : I = [1_3]
```

*** erreur (16) du type ";" attendu ***

```
fam : I = [1_3]
^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM013.PAS"

```
fam : I = [5_3];
```

*** erreur (6) du type bornes menant à un intervalle invalide ***

```
fam : I = [5_3];
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM014.PAS"

```
fam : I = (I, maman);
```

*** erreur (37) du type nom deja utilise ***

```
fam : I = (I, maman);
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM015.PAS"

```
fam : I = (cst, maman);

        *** erreur (42) du type mot reserve ***

fam : I = (cst, maman);
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM016.PAS"

```
fam : I = bruxelles,namur;

     *** erreur (15) du type "(" ou "[" attendu ***

fam : I = bruxelles,namur;
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM017.PAS"

```
fam : I = (bruxelles,bruxelles);

      *** erreur (37) du type nom deja utilise ***

fam : I = (bruxelles,bruxelles);
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM018.PAS"

```
fam : I = (bruxelles

        *** erreur (22) du type "," attendu ***

fam : I = (bruxelles
^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM019.PAS"

```
fam : I = ();
```

*** erreur (22) du type "," attendu ***

```
fam : I = ();
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM020.PAS"

```
fam : I = (namur,liege;
```

*** erreur (12) du type ")" attendu ***

```
fam : I = (namur,liege;
                       ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM021.PAS"

```
fam : I = (namur,liege)
```

*** erreur (16) du type ";" attendu ***

```
fam : I = (namur,liege)
^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM022.PAS"

```
fam : I = (namur,liege);
      I = [1_3];

      *** erreur (37) du type nom deja utilise ***

fam : I = (namur,liege);
      I = [1_3];
        ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM023.PAS"

```
fam : f1 = (namur,liege);
      f2 = [1_3];
      f3 = (unpointcinq,troispointcinq);
      f4 = [-10_10];
      f5 = (x,y);
      f6 = (x,w);
      f7 = [-2_2];
      f8 = [1_5];
      f9 = (a,b);
      f10 = [-3_0];
      f11 = [-4_-1];
      f12 = [1_2];
      f13 = [-2_2];
      f14 = [1_5];
      f15 = [1_3];
      f16 = [10_12];

  *** erreur (58) du type trop de noms de familles ***

      f15 = [1_3];
      f16 = [10_12];
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM024.PAS"

```
fam : I = (namur,liege,bruxelles,gand,anvers,bruges);
      J = (wavre,lln,ottignies,arlon,malines);
```

*** erreur (31) du type trop de chaines alphanumeriques **

```
fam : I = (namur,liege,bruxelles,gand,anvers,bruges);
      J = (wavre,lln,ottignies,arlon,malines);
                                              ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM025.PAS"

```
fam : i = [1,3];
```

*** erreur (56) du type entier ou "_" attendu ***

```
fam : i = [1,3];
          ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM026.PAS"

```
fam : cst = [1_3];
```

*** erreur (42) du type mot reserve ***

```
fam : cst = [1_3];
            ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM027.PAS"

```
fam . I = [1_10];
```

*** erreur (3) du type nom de paragraphe attendu ***

```
fam . I = [1_10];
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM028.PAS"

```
cst : I = [1_10];
```

*** erreur (16) du type ";" attendu ***

```
cst : I = [1_10];
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM029.PAS"

```
cst : I = (bruxelles,liege);
```

*** erreur (16) du type ";" attendu ***

```
cst : I = (bruxelles,liege);
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM030.PAS"

```
som : I = (bruxelles,liege);
```

*** erreur (3) du type nom de paragraphe attendu ***

```
som : I = (bruxelles,liege);
```

```
TRAITEMENT DU FICHIER "B:\FAMILLE\FAM031.PAS"

equ : I = (bruxelles,liege);

   *** erreur (43) du type "min" ou "max" attendu ***

equ : I = (bruxelles,liege);
      ^

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM032.PAS"

fam : I = [1_70000];

  *** erreur ( 7) du type borne superieure invalide ***

fam : I = [1_70000];
              ^

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM033.PAS"

fam : impair = (1,3,5,7);

        *** erreur ( 1) du type nom invalide ***

fam : impair = (1,3,5,7);
               ^

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM034.PAS"

fam : xyz = [-70000_1];

  *** erreur ( 8) du type borne inferieure invalide ***

fam : xyz = [-70000_1];
                ^
```

TRAITEMENT DU FICHIER "B:\FAMILLE\FAM035.PAS"

```
fam, i = [1_3];
```

*** erreur (3) du type nom de paragraphe attendu ***

```
fam, i = [1_3];
```

TESTS SUR LES CONSTANTES

TRAITEMENT DU FICHIER "B:\CONSTANT\CST000.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C0 = 20;
      C1(I) = 1,2,3;
      C2(I,J) = 1,2,3,4,5,6;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I, C(I) = 1;
      #I, Cbis1(I,J) = 1,2;
      #J, Cbis2(I,J) = 1,2,3;
      #I, #J, Cbis12(I,J) = 1;
      #I, Cter1(I,J,K) = 1,2,3,4,5,6,7,8;
      #J, Cter2(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12;
      #K, Cter3(I,J,K) = 1,2,3,4,5,6;
      #I, #J, Cter12(i,j,k) = 1,2,3,4;
      #I, #K, Cter13(i,j,k) = 1,2;
      #K, #J, Cter23(i,j,k) = 1,2,3;
      #i, #j, #k, Cter123(i,j,k) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

15 constantes :

  1 c0
  # :    1 ,  2 ,  3
fam :    1 ,  2 ,  3
 20.0000

  2 c1
  # :    1 ,  2 ,  3
fam :    1 i,  2 ,  3
  1.0000  2.0000  3.0000
```

```
  3 c2
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3
  1.0000   2.0000   3.0000   4.0000   5.0000   6.0000

  4 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k

  1.0000   2.0000   3.0000   4.0000   5.0000   6.0000
  7.0000   8.0000   9.0000  10.0000  11.0000  12.0000
 13.0000  14.0000  15.0000  16.0000  17.0000  18.0000
 19.0000  20.0000  21.0000  22.0000  23.0000  24.0000

  5 c
  # :    1 i,   2 ,   3
fam :    1 i,   2 ,   3
  1.0000   1.0000   1.0000

  6 cbis1
  # :    1 i,   2 ,   3
fam :    1 i,   2 j,   3
  1.0000   2.0000   1.0000   2.0000   1.0000   2.0000

  7 cbis2
  # :    1 j,   2 ,   3
fam :    1 i,   2 j,   3
  1.0000   1.0000   2.0000   2.0000   3.0000   3.0000

  8 cbis12
  # :    1 i,   2 j,   3
fam :    1 i,   2 j,   3
  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000

  9 cter1
  # :    1 i,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000   2.0000   3.0000   4.0000   5.0000   6.0000
  7.0000   8.0000   1.0000   2.0000   3.0000   4.0000
  5.0000   6.0000   7.0000   8.0000   1.0000   2.0000
  3.0000   4.0000   5.0000   6.0000   7.0000   8.0000

 10 cter2
  # :    1 j,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000   2.0000   3.0000   4.0000   1.0000   2.0000
  3.0000   4.0000   5.0000   6.0000   7.0000   8.0000
  5.0000   6.0000   7.0000   8.0000   9.0000  10.0000
 11.0000  12.0000   9.0000  10.0000  11.0000  12.0000

 11 cter3
  # :    1 k,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000   1.0000   1.0000   1.0000   2.0000   2.0000
  2.0000   2.0000   3.0000   3.0000   3.0000   3.0000
  4.0000   4.0000   4.0000   4.0000   5.0000   5.0000
  5.0000   5.0000   6.0000   6.0000   6.0000   6.0000
```

```
 12 cter12
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  1.0000  2.0000
  3.0000  4.0000  1.0000  2.0000  3.0000  4.0000
  1.0000  2.0000  3.0000  4.0000  1.0000  2.0000
  3.0000  4.0000  1.0000  2.0000  3.0000  4.0000

 13 cter13
  # :    1 i,  2 k,  3
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000
  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000
  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000
  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000

 14 cter23
  # :    1 k,  2 j,  3
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000
  2.0000  2.0000  2.0000  2.0000  3.0000  3.0000
  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000

 15 cter123
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST001.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C0 = 20;
      C0 = 10;
      C1(I) = 1,2,3;
      C1(2) = 0;
      C2(I,J) = 1,2,3,4,5,6;
      C2(2,2) = 0;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      C3(2,2,2) = 0;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 4 constantes :

  1 c0
  # :    1 ,  2 ,  3
fam :    1 ,  2 ,  3
 10.0000

  2 c1
  # :    1 ,  2 ,  3
fam :    1 i,  2 ,  3
  1.0000  0.0000  3.0000

  3 c2
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3
  1.0000  2.0000  3.0000  0.0000  5.0000  6.0000

  4 c3
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  5.0000  6.0000
  7.0000  8.0000  9.0000 10.0000 11.0000 12.0000
 13.0000  0.0000 15.0000 16.0000 17.0000 18.0000
 19.0000 20.0000 21.0000 22.0000 23.0000 24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST002.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];
```

```
cst : C0 = 20;
      C1(I) = 1,2,3;
      #I, C1(I) = 0;
      C2(I,J) = 1,2,3,4,5,6;
      #I,#J, C2(I,J) = 0;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I,#J,#K,C3(I,J,K) = 0;
fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

```
paragraphe "constantes" traite

 4 constantes :

  1 c0
  # :    1 ,   2 ,   3
fam :    1 ,   2 ,   3
 20.0000

  2 c1
  # :    1 ,   2 ,   3
fam :    1 i,   2 ,   3
  0.0000   0.0000   0.0000

  3 c2
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000

  4 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST003.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C2(I,J) = 1,2,3,4,5,6;
      #I, C2(I,2) = 0;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I, C3(I,2,2) = 0;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 2 constantes :

  1 c2
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3
  1.0000   0.0000   3.0000   0.0000   5.0000   0.0000

 2 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000   2.0000   3.0000   4.0000   5.0000   0.0000
  7.0000   8.0000   9.0000  10.0000  11.0000  12.0000
 13.0000   0.0000  15.0000  16.0000  17.0000  18.0000
 19.0000  20.0000  21.0000   0.0000  23.0000  24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST004.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C2(I,J) = 1,2,3,4,5,6;
      #J, C2(2,J) = 0;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #J, C3(2,J,2) = 0;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 2 constantes :

  1 c2
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3
  1.0000  2.0000  0.0000  0.0000  5.0000  6.0000

  2 c3
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  5.0000  6.0000
  7.0000  8.0000  9.0000  0.0000 11.0000 12.0000
 13.0000  0.0000 15.0000 16.0000 17.0000 18.0000
 19.0000 20.0000 21.0000 22.0000 23.0000 24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST005.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #K, C3(2,2,K) = 0;

fin :
```

paragraphe "familles" traite

3 familles :

1 i
par intervalle :
de 1 a 3

2 j
par intervalle :
de i a 2

3 k
par intervalle :
de 1 a 4

paragraphe "constantes" traite

1 constantes :

1 c3

# :	1 ,	2 ,	3		
fam :	1 i,	2 j,	3 k		
1.0000	2.0000	3.0000	4.0000	5.0000	6.0000
7.0000	8.0000	9.0000	10.0000	11.0000	12.0000
0.0000	0.0000	0.0000	0.0000	17.0000	18.0000
19.0000	20.0000	21.0000	22.0000	23.0000	24.0000

TRAITEMENT DU FICHIER "B:\CONSTANT\CST006.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I,#J, C3(I,J,2) = 0;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000  0.0000  3.0000  4.0000  5.0000  0.0000
  7.0000  8.0000  9.0000  0.0000 11.0000 12.0000
 13.0000  0.0000 15.0000 16.0000 17.0000  0.0000
 19.0000 20.0000 21.0000  0.0000 23.0000 24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST007.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I,#K, C3(I,2,K) = 0;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

```
paragraphe "constantes" traite

 1 constantes :

  1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000  2.0000  3.0000  4.0000  0.0000  0.0000
  0.0000  0.0000  9.0000 10.0000 11.0000 12.0000
  0.0000  0.0000  0.0000  0.0000 17.0000 18.0000
 19.0000 20.0000 21.0000 22.0000 23.0000 24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST008.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #J,#K, C3(2,J,K) = 0;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
  1.0000  2.0000  3.0000  4.0000  5.0000  6.0000
  7.0000  8.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000 17.0000 18.0000
 19.0000 20.0000 21.0000 22.0000 23.0000 24.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST009.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C2(I,J) = 1,2,3,4,5,6;
      #I, C2(I,J) = 10,20;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I, C3(I,J,K) =10,20,30,40,50,60,70,80;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 2 constantes :

  1 c2
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3
 10.0000 20.0000 10.0000 20.0000 10.0000 20.0000

  2 c3
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3 k
 10.0000 20.0000 30.0000 40.0000 50.0000 60.0000
 70.0000 80.0000 10.0000 20.0000 30.0000 40.0000
 50.0000 60.0000 70.0000 80.0000 10.0000 20.0000
 30.0000 40.0000 50.0000 60.0000 70.0000 80.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST010.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C2(I,J) = 1,2,3,4,5,6;
      #J, C2(I,J) = 10,20,30;
      C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #J, C3(I,J,K) =
10,20,30,40,50,60,70,80,90,100,110,120;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 2 constantes :

  1 c2
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3
 10.0000 10.0000 20.0000 20.0000 30.0000 30.0000

  2 c3
  # :    1 ,  2 ,  3
fam :    1 i,  2 j,  3 k
 10.0000 20.0000 30.0000 40.0000 10.0000 20.0000
 30.0000 40.0000 50.0000 60.0000 70.0000 80.0000
 50.0000 60.0000 70.0000 80.0000 90.0000 100.000
 110.000 120.000 90.0000 100.000 110.000 120.000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST011.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #K, C3(I,J,K) = 10,20,30,40,50,60;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
 10.0000 10.0000 10.0000 10.0000 20.0000
 20.0000 20.0000 20.0000 30.0000 30.0000
 30.0000 30.0000 40.0000 40.0000 40.0000
 40.0000 50.0000 50.0000 50.0000 50.0000
 60.0000 60.0000 60.0000 60.0000 60.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST012.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I,#J, C3(I,J,K) = 10,20,30,40;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
 10.0000 20.0000 30.0000 40.0000 10.0000 20.0000
 30.0000 40.0000 10.0000 20.0000 30.0000 40.0000
 10.0000 20.0000 30.0000 40.0000 10.0000 20.0000
 30.0000 40.0000 10.0000 20.0000 30.0000 40.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST013.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #I,#K, C3(I,J,K) = 10,20;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

```
paragraphe "constantes" traite

 1 constantes :

   1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
 10.0000 10.0000 10.0000 10.0000 20.0000 20.0000
 20.0000 20.0000 10.0000 10.0000 10.0000 10.0000
 20.0000 20.0000 20.0000 20.0000 10.0000 10.0000
 10.0000 10.0000 20.0000 20.0000 20.0000 20.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST014.PAS"

```
fam : I = [1_3];
      J = [1_2];
      K = [1_4];

cst : C3(I,J,K) = 1,2,3,4,5,6,7,8,9,10,11,12,
                  13,14,15,16,17,18,19,20,21,22,23,24;
      #J,#K, C3(I,J,K) = 10,20,30;

fin :


paragraphe "familles" traite

 3 familles :

   1 i
par intervalle :
de      1 a      3

   2 j
par intervalle :
de      1 a      2

   3 k
par intervalle :
de      1 a      4

paragraphe "constantes" traite

 1 constantes :

   1 c3
  # :    1 ,   2 ,   3
fam :    1 i,   2 j,   3 k
 10.0000 10.0000 10.0000 10.0000 10.0000 10.0000
 10.0000 10.0000 20.0000 20.0000 20.0000 20.0000
 20.0000 20.0000 20.0000 20.0000 30.0000 30.0000
 30.0000 30.0000 30.0000 30.0000 30.0000 30.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST015.PAS"

```
c = 20;

*** erreur ( 3) du type nom de paragraphe attendu ***

c = 20;
  ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST016.PAS"

```
cst c = 20;

  *** erreur ( 3) du type nom de paragraphe attendu ***

cst c = 20;
      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST017.PAS"

```
cst . c = 20;

  *** erreur ( 3) du type nom de paragraphe attendu ***

cst . c = 20;
      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST018.PAS"

```
fam : c = 20;

     *** erreur (15) du type "(" ou "[" attendu ***

fam : c = 20;
        ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST019.PAS"

```
som : c = 20;
```

*** erreur (3) du type nom de paragraphe attendu ***

```
som : c = 20;
    ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST020.PAS"

```
equ : c = 20;
```

*** erreur (43) du type "min" ou "max" attendu ***

```
equ : c = 20;
      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST021.PAS"

```
cst :  c = 1,2;
```

*** erreur (11) du type enumeration trop longue ***

```
cst :  c = 1,2;
           ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST022.PAS"

```
cst :  c = ;
```

*** erreur (27) du type type de valeur invalide ***

```
cst :  c = ;
           ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST023.PAS"

```
cst :  c =
```

*** erreur (16) du type ";" attendu ***

```
cst :  c =
^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST024.PAS"

```
fam : i = [1_3];
cst : c(i) = 1,2;
```

paragraphe "familles" traite

```
 1 familles :

  1 i
par intervalle :
de     1 a     3
```

*** erreur (29) du type enumeration insuffisante ***

```
fam : i = [1_3];
cst : c(i) = 1,2;
              ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST025.PAS"

```
fam : i = [1_3];
cst : c(i) = 1,2,3,4;
```

```
paragraphe "familles" traite

 1 familles :

   1 i
par intervalle :
de      1 a      3
```

*** erreur (11) du type enumeration trop longue ***

```
fam : i = [1_3];

cst : c(i) = 1,2,3,4;
                  ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST026.PAS"

```
cst : c() = 1;
```

*** erreur (20) du type nom de famille invalide ***

```
cst : c() = 1;
        ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST027.PAS"

```
cst : c(1) = 1,2,3;
```

*** erreur (20) du type nom de famille invalide ***

```
cst : c(1) = 1,2,3;
        ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST028.PAS"

```
cst : c 25;
```

*** erreur (23) du type "(" ou "=" attendu ***

```
cst : c 25;
      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST029.PAS"

```
cst : c : 25;
```

*** erreur (23) du type "(" ou "=" attendu ***

```
cst : c : 25;
      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST030.PAS"

```
fam : i = [1_3];
cst : #i, c(i) = 1,2;

paragraphe "familles" traite
 1 familles :

  1 i
par intervalle :
de     1 a     3
```

*** erreur (11) du type enumeration trop longue ***

```
fam : i = [1_3];
cst : #i, c(i) = 1,2;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST031.PAS"

```
fam : i = [1_3];

cst : #i, c(i) = ;


paragraphe "familles" traite

 1 familles :

  1 i
par intervalle :
de     1 a     3


*** erreur (27) du type type de valeur invalide ***

fam : i = [1_3];

cst : #i, c(i) = ;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST032.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i,j) = ;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


   *** erreur (27) du type type de valeur invalide ***

      j = [1_2];

cst : #i, c(i,j) = ;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST033.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i,j) = 1,2,3;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

**** erreur (11) du type enumeration trop longue ****

```
      j = [1_2];

cst : #i, c(i,j) = 1,2,3;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST034.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #j, c(i,j) = 1,2;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

**** erreur (29) du type enumeration insuffisante ****

```
      j = [1_2];

cst : #j, c(i,j) = 1,2;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST035.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #j, c(i,j) = 1,2,3,4;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

**** erreur (11) du type enumeration trop longue ****

```
      j = [1_2];
cst : #j, c(i,j) = 1,2,3,4;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST036.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #j,#i, c(i,j) = 1,2,3,4,5,6,7;
```

```
paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


   *** erreur (11) du type enumeration trop longue ***


      j = [1_2];

cst : #j,#i, c(i,j) = 1,2,3,4,5,6,7;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST037.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #j,#i, c(i,j) =;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


   *** erreur (27) du type type de valeur invalide ***


      j = [1_2];

cst : #j,#i, c(i,j) =;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST038.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j,#i, c(k) = 1;
```

paragraphe "familles" traite

```
 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

**** erreur (26) du type declaration incoherente ****

```
      k = [1_4];

cst : #j,#i, c(k) = 1;
                    ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST039.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j,#i,#k, c(i,j,k) = 1,2;
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

   *** erreur (11) du type enumeration trop longue ***


      k = [1_4];

cst : #j,#i,#k, c(i,j,k) = 1,2;
                           ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST040.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j,#i,#k, c(i,j,k) = ;

paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

*** erreur (27) du type type de valeur invalide ***

```
      k = [1_4];

cst : #j,#i,#k, c(i,j,k) = ;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST041.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j,#i, c(i,j,k) = 1,2;
```

paragraphe "familles" traite

3 familles :

```
  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

*** erreur (29) du type enumeration insuffisante ***

```
      k = [1_4];

cst : #j,#i, c(i,j,k) = 1,2;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST042.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j,#i, c(i,j,k) = 1,2,3,4,5;
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

**** erreur (11) du type enumeration trop longue ****

```
      k = [1_4];

cst : #j,#i, c(i,j,k) = 1,2,3,4,5;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST043.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j, c(i,j,k) = 1,2,3,4,5;

paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

*** erreur (29) du type enumeration insuffisante ***

```
      k = [1_4];

cst : #j, c(i,j,k) = 1,2,3,4,5;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST044.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #j, c(i,j,k) = 1,2,3,4,5,6,7,8,9,10,11,12,13;
```

paragraphe "familles" traite

3 familles :

1 i
par intervalle :
de 1 a 3

2 j
par intervalle :
de 1 a 2

3 k
par intervalle :
de 1 a 4

*** erreur (11) du type enumeration trop longue ***

```
      k = [1_4];

cst : #j, c(i,j,k) = 1,2,3,4,5,6,7,8,9,10,11,12,13;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST045.PAS"

```
fam : i = [1_3];

cst : c[i] = 1,2,3;
```

```
paragraphe "familles" traite

 1 familles :

  1 i
par intervalle :
de     1 a     3

     *** erreur (23) du type "(" ou "=" attendu ***

fam : i = [1_3];

cst : c[i] = 1,2,3;
       ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST046.PAS"

```
fam : i = [1_3];

cst : c(i = 1,2,3;

paragraphe "familles" traite

 1 familles :

  1 i
par intervalle :
de     1 a     3

     *** erreur (19) du type "," ou ")" attendu ***

fam : i = [1_3];

cst : c(i = 1,2,3;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST047.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : c(i.j) = 1,2,3,4,5,6;
```

```
paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (20) du type nom de famille invalide ***

```
      j = [1_2];

cst : c(i.j) = 1,2,3,4,5,6;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST048.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c = 1;

paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (26) du type declaration incoherente ***

```
      j = [1_2];

cst : #i, c = 1;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST049.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #k, c(i,j) = 1,2,3,4,5,6;

paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

**** erreur (20) du type nom de famille invalide ****

```
      j = [1_2];

cst : #k, c(i,j) = 1,2,3,4,5,6;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST050.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #k, c(i,j) = 1,2,3,4,5,6;

paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

```
  3 k
par intervalle :
de     1 a     4
```

*** erreur (26) du type declaration incoherente ***

```
      k = [1_4];

cst : #k, c(i,j) = 1,2,3,4,5,6;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST051.PAS"

```
fam : i = [1_3];

cst : #i, c(i) = 1;
      c(4) = 0;
```

paragraphe "familles" traite

```
 1 familles :

  1 i
par intervalle :
de     1 a     3
```

*** erreur (39) du type valeur non comprise dans les bornes ***

```
cst : #i, c(i) = 1;
      c(4) = 0;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST052.PAS"

```
fam : ville = (bruxelles,namur);

cst : #ville, c(ville) = 1;
      c(wavre) = 0;
```

```
paragraphe "familles" traite

 1 familles :

  1 ville
par enumeration :
bruxelles namur


*** erreur (41) du type nom non existant dans
l'enumeration ***


cst : #ville, c(ville) = 1;
      c(wavre) = 0;
            ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST053.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, #j, c(i,j) = 1;
      c(2,3) = 0;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


*** erreur (39) du type valeur non comprise dans les
bornes ***


cst : #i, #j, c(i,j) = 1;
      c(2,3) = 0;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST054.PAS"

```
fam : i = [1_3];
      ville = (bruxelles,namur);

cst : #i, #ville, c(i,ville) = 1;
      c(2,wavre) = 0;
```

paragraphe "familles" traite

```
 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 ville
par enumeration :
bruxelles namur
```

*** *erreur (41) du type nom non existant dans l'enumeration* ***

```
cst : #i, #ville, c(i,ville) = 1;
      c(2,wavre) = 0;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST055.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      ville = (bruxelles,namur);

cst : #i, #j, #k, c(i,j,k) = 1;
      #i, #j, c(i,j,5) = 0;
```

paragraphe "familles" traite

```
 4 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

```
  3 k
par intervalle :
de     1 a     4

  4 ville
par enumeration :
bruxelles namur


*** erreur (39) du type valeur non comprise dans les
bornes ***


cst : #i, #j, #k, c(i,j,k) = 1;
      #i, #j, c(i,j,5) = 0;
                       ^
```

```
TRAITEMENT DU FICHIER "B:\CONSTANT\CST056.PAS"
```

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      ville = (bruxelles,namur);

cst : #i, #ville, #k, c(i,ville,k) = 1;
      #i, #k, c(i,wavre,k) = 0;


paragraphe "familles" traite

 4 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

  4 ville
par enumeration :
bruxelles namur
```

```
*** erreur (41) du type nom non existant dans
l'enumeration ***


cst : #i, #ville, #k, c(i,ville,k) = 1;
      #i, #k, c(i,wavre,k) = 0;
                           ^
```

```
TRAITEMENT DU FICHIER "B:\CONSTANT\CST057.PAS"
```

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      ville = (bruxelles,namur);

cst : #i, #j, #k, c(i,j,k) = 1;
      #i, #j, #k, c(i,j,4) = 0;


paragraphe "familles" traite

 4 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

  4 ville
par enumeration :
bruxelles namur


   *** erreur (26) du type declaration incoherente ***


cst : #i, #j, #k, c(i,j,k) = 1;
      #i, #j, #k, c(i,j,4) = 0;
                           ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST058.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      ville = (bruxelles,namur);

cst : #i, #j, c(i,j) = 1;
      #i, #j, c(2,j) = 0;
```

paragraphe "familles" traite

4 familles :

1 i
par intervalle :
de 1 a 3

2 j
par intervalle :
de 1 a 2

3 k
par intervalle :
de 1 a 4

4 ville
par enumeration :
bruxelles namur

*** erreur (26) du type declaration incoherente ***

```
cst : #i, #j, c(i,j) = 1;
      #i, #j, c(2,j) = 0;
                  ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST059.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i) = 1;
      #i, c(i) = 0,2;
```

```
paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


   *** erreur (11) du type enumeration trop longue ***


cst : #i, c(i) = 1;
      #i, c(i) = 0,2;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST060.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i) = 1;
      c(2) = 0,2;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2


   *** erreur (11) du type enumeration trop longue ***


cst : #i, c(i) = 1;
      c(2) = 0,2;
                ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST061.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, #j, c(i,j) = 1;
      #i, c(i,j) = 0;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

**** erreur (29) du type enumeration insuffisante ****

```
cst : #i, #j, c(i,j) = 1;
      #i, c(i,j) = 0;
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST062.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i) = 1;
      c(i) = 2,3,4,5;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (11) du type enumeration trop longue ***

```
cst : #i, c(i) = 1;
      c(i) = 2,3,4,5;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST063.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i) = 1;
      c(i) = 2,3;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (29) du type enumeration insuffisante ***

```
cst : #i, c(i) = 1;
      c(i) = 2,3;
               ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST064.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, #j, c(i,j) = 0;
      #i, c(i,j) = 1;
```

```
paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

   *** erreur (29) du type enumeration insuffisante ***


cst : #i, #j, c(i,j) = 0;
      #i, c(i,j) = 1;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST065.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, #j, c(i,j) = 0;
      #i, c(i,j) = 1,2,3;


paragraphe "familles" traite

 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

    *** erreur (11) du type enumeration trop longue ***


cst : #i, #j, c(i,j) = 0;
      #i, c(i,j) = 1,2,3;
                   ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST066.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, c(i) = 0;
      #i, c(i,j) = 1,2;
```

paragraphe "familles" traite

```
 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (26) du type declaration incoherente ***

```
cst : #i, c(i) = 0;
      #i, c(i,j) = 1,2;
             ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST067.PAS"

```
fam : i = [1_3];
      j = [1_2];

cst : #i, #j, c(i,j) = 0;
      #i, #j, c(j,i) = 1;
```

paragraphe "familles" traite

```
 2 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2
```

*** erreur (26) du type declaration incoherente ***

```
cst : #i, #j, c(i,j) = 0;
      #i, #j, c(j,i) = 1;
                ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST068.PAS"

```
fam : i = [1_3];

cst : c = 0;
      #i, c(i) = 1;
```

paragraphe "familles" traite

```
 1 familles :

   1 i
par intervalle :
de      1 a      3
```

*** erreur (26) du type declaration incoherente ***

```
cst : c = 0;
      #i, c(i) = 1;
            ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST069.PAS"

```
fam : i = [1_3];

cst : #i c(i) = 1;
```

paragraphe "familles" traite

```
 1 familles :

   1 i
par intervalle :
de      1 a      3
```

**** erreur (22) du type "," attendu ****

```
fam : i = [1_3];

cst : #i c(i) = 1;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST070.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, c(i,j,2) = 1,2;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :
  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  0.0000   1.0000   0.0000   0.0000   0.0000   2.0000
  0.0000   0.0000   0.0000   1.0000   0.0000   0.0000
  0.0000   2.0000   0.0000   0.0000   0.0000   1.0000
  0.0000   0.0000   0.0000   2.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST071.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, c(i,2,k) = 1,2,3,4;

fin :
```

paragraphe "familles" traite

3 familles :

1 i
par intervalle :
de 1 a 3

2 j
par intervalle :
de 1 a 2

3 k
par intervalle :
de 1 a 4

paragraphe "constantes" traite

1 constantes :

1 c

# :	1 i,	2 j,	3 k		
fam :	1 i,	2 j,	3 k		
0.0000	0.0000	0.0000	0.0000	1.0000	2.0000
3.0000	4.0000	0.0000	0.0000	0.0000	0.0000
1.0000	2.0000	3.0000	4.0000	0.0000	0.0000
0.0000	0.0000	1.0000	2.0000	3.0000	4.0000

TRAITEMENT DU FICHIER "B:\CONSTANT\CST072.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, c(2,j,k) = 1,2,3,4;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   1.0000   2.0000   3.0000   4.0000
  1.0000   2.0000   3.0000   4.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST073.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, c(i,j,2) = 1,2,3;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4
```

```
paragraphe "constantes" traite

 1 constantes :

   1 c
   # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
   0.0000   1.0000   0.0000   0.0000   0.0000   1.0000
   0.0000   0.0000   0.0000   2.0000   0.0000   0.0000
   0.0000   2.0000   0.0000   0.0000   0.0000   3.0000
   0.0000   0.0000   0.0000   3.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST074.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(2,j,k) = 1,2;

fin :


paragraphe "familles" traite

 3 familles :

   1 i
par intervalle :
de     1 a     3

   2 j
par intervalle :
de     1 a     2

   3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

   1 c
   # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   1.0000   1.0000   1.0000   1.0000
   2.0000   2.0000   2.0000   2.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST075.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(i,2,k) = 1,2,3;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par intervalle :
de     1 a     3

  2 j
par intervalle :
de     1 a     2

  3 k
par intervalle :
de     1 a     4

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  0.0000   0.0000   0.0000   0.0000   1.0000   1.0000
  1.0000   1.0000   0.0000   0.0000   0.0000   0.0000
  2.0000   2.0000   2.0000   2.0000   0.0000   0.0000
  0.0000   0.0000   3.0000   3.0000   3.0000   3.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST076.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, #j, #k, c(i,j,k) = 1;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST077.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #i, #j, c1(i,j,sambre) = 1;
      #i, #j, c2(i,j) = 0;
      c2(i,liege) = 1,2,3;
      #i, c3(i) = 0;
      #i, c3(i) = 1;
      c3(bxl) = 2;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege
```

```
  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 3 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  1.0000  0.0000  0.0000  0.0000  1.0000
  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000  1.0000
  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  0.0000  1.0000  0.0000  2.0000  0.0000  3.0000

  3 c3
  # :    1 i,  2 ,  3
fam :    1 i,  2 ,  3
  1.0000  2.0000  1.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST078.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #i, #k, c1(i,liege,k) = 1;
      #i, #j, c2(i,j) = 0;
      c2(bxl,j) = 1,2;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois
```

```
paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  0.0000  0.0000  0.0000  1.0000  1.0000
  1.0000  1.0000  0.0000  0.0000  0.0000  0.0000
  1.0000  1.0000  1.0000  1.0000  0.0000  0.0000
  0.0000  0.0000  1.0000  1.0000  1.0000  1.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  0.0000  0.0000  1.0000  2.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST079.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #j, #k, c1(bxl,j,k) = 1;
      #i, #j, c2(i,j) = 0;
      c2(bxl,liege) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois
```

```
paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  1.0000  1.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST080.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, #k, c(i,j,k) = 1,2,3;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000
  2.0000  2.0000  2.0000  2.0000  3.0000  3.0000
  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST081.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, #k, c(i,j,k) = 1,2;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000
  2.0000  2.0000  1.0000  1.0000  1.0000  1.0000
  2.0000  2.0000  2.0000  2.0000  1.0000  1.0000
  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST082.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #i, #j, c1(i,j,k) = 1,2,3,4;
      #i, #j, c2(i,j) = 0;
      #i, #j, c2(i,j) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  1.0000  2.0000
  3.0000  4.0000  1.0000  2.0000  3.0000  4.0000
  1.0000  2.0000  3.0000  4.0000  1.0000  2.0000
  3.0000  4.0000  1.0000  2.0000  3.0000  4.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST083.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, c(i,liege,sambre) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  0.0000  0.0000  0.0000  0.0000  1.0000
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  1.0000  0.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST084.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #i, c(i,liege,k) = 1,2,3,4;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  0.0000   0.0000   0.0000   0.0000   1.0000   2.0000
  3.0000   4.0000   0.0000   0.0000   0.0000   0.0000
  1.0000   2.0000   3.0000   4.0000   0.0000   0.0000
  0.0000   0.0000   1.0000   2.0000   3.0000   4.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST085.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #i, c1(i,j,sambre) = 1,2;
      #i, #j, c2(i,j) = 0;
      #i, c2(i,liege) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege
```

```
  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  1.0000  0.0000  0.0000  0.0000  2.0000
  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  2.0000  0.0000  0.0000  0.0000  1.0000
  0.0000  0.0000  0.0000  2.0000  0.0000  0.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  0.0000  1.0000  0.0000  1.0000  0.0000  1.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST086.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, c(i,j,k) = 1,2,3,4,5,6,7,8,9,10,11,12;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois
```

```
paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  1.0000  2.0000
  3.0000  4.0000  5.0000  6.0000  7.0000  8.0000
  5.0000  6.0000  7.0000  8.0000  9.0000 10.0000
 11.0000 12.0000  9.0000 10.0000 11.0000 12.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST087.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, c(bxl,j,sambre) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  1.0000  0.0000  0.0000
  0.0000  1.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST088.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #j, c(i,j,sambre) = 1,2,3;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  1.0000  0.0000  0.0000  0.0000  1.0000
  0.0000  0.0000  0.0000  2.0000  0.0000  0.0000
  0.0000  2.0000  0.0000  0.0000  0.0000  3.0000
  0.0000  0.0000  0.0000  3.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST089.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      #j, c1(bxl,j,k) = 1,2,3,4;
      #i, #j, c2(i,j) = 0;
      #j, c2(bxl,j) = 1;

fin :
```

```
paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
  0.0000  0.0000  1.0000  2.0000  3.0000  4.0000
  1.0000  2.0000  3.0000  4.0000  0.0000  0.0000
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  0.0000  0.0000  1.0000  1.0000  0.0000  0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST090.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(i,j,k) = 1,2,3,4,5,6;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre
```

```
  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  1.0000  1.0000  1.0000  2.0000  2.0000
  2.0000  2.0000  3.0000  3.0000  3.0000  3.0000
  4.0000  4.0000  4.0000  4.0000  5.0000  5.0000
  5.0000  5.0000  6.0000  6.0000  6.0000  6.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST091.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(bxl,liege,k) = 1;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois
```

```
paragraphe "constantes" traite

 1 constantes :

   1 c
   # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
   1.0000   1.0000   1.0000   1.0000   0.0000   0.0000
   0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST092.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(i,liege,k) = 1,2,3;

fin :


paragraphe "familles" traite

 3 familles :

   1 i
par enumeration :
namur bxl wavre

   2 j
par enumeration :
bxl liege

   3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

   1 c
   # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
   0.0000   0.0000   0.0000   0.0000   1.0000   1.0000
   1.0000   1.0000   0.0000   0.0000   0.0000   0.0000
   2.0000   2.0000   2.0000   2.0000   0.0000   0.0000
   0.0000   0.0000   3.0000   3.0000   3.0000   3.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST093.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c(i,j,k) = 0;
      #k, c(bxl,j,k) = 1,2;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :    1 i,   2 j,   3 k
fam :    1 i,   2 j,   3 k
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
  0.0000   0.0000   1.0000   1.0000   1.0000   1.0000
  2.0000   2.0000   2.0000   2.0000   0.0000   0.0000
  0.0000   0.0000   0.0000   0.0000   0.0000   0.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST094.PAS"

```
fam : i = (namur,bxl,wavre);
      j = (bxl,liege);
      k = (meuse,sambre,escaut,semois);

cst : #i, #j, #k, c1(i,j,k) = 0;
      c1(i,j,k) =
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
                  20,21,22,23,24;
      #i, #j, c2(i,j) = 0;
      c2(i,j) = 1,2,3,4,5,6;

fin :


paragraphe "familles" traite

 3 familles :

  1 i
par enumeration :
namur bxl wavre

  2 j
par enumeration :
bxl liege

  3 k
par enumeration :
meuse sambre escaut semois

paragraphe "constantes" traite

 2 constantes :

  1 c1
  # :    1 i,  2 j,  3 k
fam :    1 i,  2 j,  3 k
  1.0000  2.0000  3.0000  4.0000  5.0000  6.0000
  7.0000  8.0000  9.0000 10.0000 11.0000 12.0000
 13.0000 14.0000 15.0000 16.0000 17.0000 18.0000
 19.0000 20.0000 21.0000 22.0000 23.0000 24.0000

  2 c2
  # :    1 i,  2 j,  3
fam :    1 i,  2 j,  3
  1.0000  2.0000  3.0000  4.0000  5.0000  6.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST095.PAS"

```
cst : c = 1;
      c = 2;
      c = 3;
      c = 4;

fin :

paragraphe "constantes" traite

 1 constantes :

  1 c
  # :   1 ,  2 ,  3
fam :   1 ,  2 ,  3
  4.0000
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST096.PAS"

```
cst : c1 = 1;
      c2 = 2;
      c3 = 3;
      c4 = 4;
      c5 = 5;
      c6 = 6;
      c7 = 7;
      c8 = 8;
      c9 = 9;
      c10 = 10;
      c11 = 11;
      c12 = 12;
      c13 = 13;
      c14 = 14;
      c15 = 15;
      c16 = 16;

 *** erreur (59) du type trop de noms de constantes ***

      c15 = 15;
      c16 = 16;
          ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST097.PAS"

```
cst, c = 2;

  *** erreur ( 3) du type nom de paragraphe attendu ***

cst, c = 2;
    ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST098.PAS"

```
cst : c = abc;

   *** erreur (27) du type type de valeur invalide ***

cst : c = abc;
           ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST099.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      l = [1_5];

cst : c(i,j,k,l) = 1,2,3,...;

       *** erreur (21) du type trop d'indices ***

      l = [1_5];

cst : c(i,j,k,l) = 1,2,3,...;
                 ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST100.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      l = [1_5];

cst : #i, #k, #l, #j, c(i,j,k,l) = 1;
```

*** erreur (21) du type trop d'indices ***

```
      l = [1_5];

cst : #i, #k, #l, #j, c(i,j,k,l) = 1;
                      ^
```

TRAITEMENT DU FICHIER "B:\CONSTANT\CST101.PAS"

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
cst : #i,j,k, c(i,j,k) = 1;
```

*** erreur (24) du type "#" attendu ***

```
      k = [1_4];
cst : #i,j,k, c(i,j,k) = 1;
            ^
```

TESTS SUR LE PARAGRAPHE "EQU:"

ET LA TRANSFORMATION DES STRUCTURES

```
equ : un, x + y = 3;
```

*** erreur (38) du type "[" attendu ***

```
****** debut du fichier *****
equ : un, x + y = 3;
        ^
```

```
equ : [x + y];
```

*** erreur (43) du type "min" ou "max" attendu ***

```
****** debut du fichier *****
equ : [x + y];
      ^
```

```
equ : opt [x + y];
```

*** erreur (43) du type "min" ou "max" attendu ***

```
****** debut du fichier *****
equ : opt [x + y];
          ^
```

```
equ : min x + y;
```

*** erreur (38) du type "[" attendu ***

```
****** debut du fichier *****
equ : min x + y;
            ^
```

```
equ : cst [x + y];
```

*** erreur (43) du type "min" ou "max" attendu ***

```
****** debut du fichier *****
equ : cst [x + y];
          ^
```

```
equ : som [x + y];
```

*** erreur (43) du type "min" ou "max" attendu ***

```
****** debut du fichier *****
equ : som [x + y];
          ^
```

```
equ : max (x + y);

      *** erreur (38) du type "[" attendu ***

****** debut du fichier *****
equ : max (x + y);
          ^



fam : i = [1_3];
equ : min [som(i) x(i)];
      un, #i, x(i) >= 1;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0

        !  cst          !       code_cst      !       val_cst
  1       0             !      0      0      0 !      0      0      0

        !  variable     !       code_var      !       val_var
  1       1             !      0      0      0 !      0      0      0

mult :
  1     1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt            val_ptt
    1     0     0 !      0     0     0 !      0     0     0
Code =    2

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst1         !      code1_cst      !     val1_cst
  1  !      0             !      0     0     0 !      0     0     0

             var1         !      code1_var      !     val1_var
  1  !      1             !      0     0     0 !      0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2         !      code2_cst      !     val2_cst
  1  !      0             !      0     0     0 !      0     0     0

    mult2
  1  1.00000
```

```
Variables :
"x"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

   ||    1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  1.0
 2 ||  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  1.0
 3 ||  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  1.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

  z =  0.00
  w =  0.00



fam : i = [1_3];
equ : min [3 * som(i) x(i)];
      un, #i, x(i) <= 1;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0

        !  cst          !       code_cst        !       val_cst
  1      0              !      0      0      0 !      0      0      0

        !  variable     !       code_var        !       val_var
  1      1              !      0      0      0 !      0      0      0

mult :
  1    3.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          !          val_ptt
    1      0      0 !      0      0      0 !      0      0      0
Code =    3
```

1er membre :

	no1_ent_som			code1_som			val1_som		
1	0	0	0	0	0	0	0	0	0

	cst1	code1_cst			val1_cst		
1	0	0	0	0	0	0	0

	var1	code1_var			val1_var		
1	1	0	0	0	0	0	0

	mult1
1	1.00000

2nd membre :

	no2_ent_som			code2_som			val2_som		
1	0	0	0	0	0	0	0	0	0

	cst2	code2_cst			val2_cst		
1	0	0	0	0	0	0	0

	mult2
1	1.00000

Variables :
"x" 1, 0, 0,

Résultats de la transformation
Tableau "C" :

‖	3.0	3.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖ !	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		1.0
2	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		1.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		1.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
equ : min [3 * som(i) 2 * x(i)];
      un, x(i >< 1) >= 4;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst        !       val_cst
  1      0              !     0     0     0 !     0     0     0

        !  variable     !       code_var        !       val_var
  1      1              !     0     0     0 !     0     0     0

mult :
  1    6.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt                val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      0     0     0 !     0     0     0 !     0     0     0

             cst1         !      code1_cst      !     val1_cst
  1  !      0              !     0     0     0 !     0     0     0

             var1         !      code1_var      !     val1_var
  1  !      1              !     1     0     0 !     1     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      0     0     0 !     0     0     0 !     0     0     0

             cst2         !      code2_cst      !     val2_cst
  1  !      0              !     0     0     0 !     0     0     0

    mult2
  1  4.00000

Variables :
"x"   1,  0,  0,

   Résultats de la transformation
Tableau "C" :

   ||   6.0  6.0  6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
equ : min [som(i <> 1) x(i)];
      un, som(i) x(i) <= 2;
fin :
```

Fonction objectif : Il faut minimiser la fonction.

no_elem	no_ent_som			code_som			val_som		
1	1	0	0	1	0	0	1	0	0

	cst	code_cst			val_cst		
1	0	0	0	0	0	0	0

	variable	code_var			val_var		
1	1	0	0	0	0	0	0

mult :

1	1.000

Equations :
equation 1 nom : "un"

no_ent_ptt			code_ptt			val_ptt		
0	0	0	0	0	0	0	0	0

Code = 3

1er membre :

	no1_ent_som			code1_som			val1_som		
1	1	0	0	0	0	0	0	0	0

	cst1	code1_cst			val1_cst		
1	0	0	0	0	0	0	0

	var1	code1_var			val1_var		
1	1	0	0	0	0	0	0

	mult1
1	1.00000

```
2nd membre :

          no2_ent_som    !     code2_som       !     val2_som
  1  !     0     0     0 !     0     0     0 !     0     0     0

           cst2          !     code2_cst       !     val2_cst
  1  !     0             !     0     0     0 !     0     0     0

    mult2
  1  2.00000

Variables :
"x"   1,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   0.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  2.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

  z =  0.00
  w =  0.00


equ : min [3 * x - 4 * y + 5 * z];
      un, 7 * y + 2 * x + 3 * y >= 6;
      deux, 9 * z + 8 * y + 5 * x >= 10;
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som       !      val_som
  1 !     0     0     0 !     0     0     0 !     0     0     0
  2 !     0     0     0 !     0     0     0 !     0     0     0
  3 !     0     0     0 !     0     0     0 !     0     0     0

        !  cst          !      code_cst       !      val_cst
  1      0              !     0     0     0 !     0     0     0
  2      0              !     0     0     0 !     0     0     0
  3      0              !     0     0     0 !     0     0     0

        !  variable     !      code_var       !      val_var
  1      1              !     0     0     0 !     0     0     0
  2      2              !     0     0     0 !     0     0     0
  3      3              !     0     0     0 !     0     0     0
```

```
mult :
  1    3.000
  2   -4.000
  3    5.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !     0     0     0 !     0     0     0 !     0     0     0
  2  !     0     0     0 !     0     0     0 !     0     0     0
  3  !     0     0     0 !     0     0     0 !     0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !     0               !     0     0     0 !     0     0     0
  2  !     0               !     0     0     0 !     0     0     0
  3  !     0               !     0     0     0 !     0     0     0

             var1          !     code1_var       !     val1_var
  1  !     2               !     0     0     0 !     0     0     0
  2  !     1               !     0     0     0 !     0     0     0
  3  !     2               !     0     0     0 !     0     0     0

    mult1
  1  7.00000
  2  2.00000
  3  3.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !     0     0     0 !     0     0     0 !     0     0     0

             cst2          !     code2_cst       !     val2_cst
  1  !     0               !     0     0     0 !     0     0     0

    mult2
  1  6.00000

Variables :
"x"    0,   0,   0,
"y"    0,   0,   0,
"z"    0,   0,   0,
   equation   2  nom : "deux"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !     0     0     0 !     0     0     0 !     0     0     0
  2  !     0     0     0 !     0     0     0 !     0     0     0
  3  !     0     0     0 !     0     0     0 !     0     0     0
```

```
          cst1          !     code1_cst          !     val1_cst
1   !     0             !     0     0     0 !     0     0     0
2   !     0             !     0     0     0 !     0     0     0
3   !     0             !     0     0     0 !     0     0     0

          var1          !     code1_var          !     val1_var
1   !     3             !     0     0     0 !     0     0     0
2   !     2             !     0     0     0 !     0     0     0
3   !     1             !     0     0     0 !     0     0     0

   mult1
1  9.00000
2  8.00000
3  5.00000

2nd membre :

          no2_ent_som   !     code2_som          !     val2_som
1   !     0     0     0 !     0     0     0 !     0     0     0

          cst2          !     code2_cst          !     val2_cst
1   !     0             !     0     0     0 !     0     0     0

   mult2
1 10.00000

Variables :
"x"   0,   0,   0,
"y"   0,   0,   0,
"z"   0,   0,   0,

   Résultats de la transformation
Tableau "C" :

  ||   3.0 -4.0  5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  2.0 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   6.0
 2 ||  5.0  8.0  9.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  10.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

   z =  0.00
   w =  0.00
```

```
equ : min [3 * x - 4 * y + 5 * z];
      un, 2 * x + 7 * y >= 0;
      deux, 8 * y + 9 * z >= 10;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
   1 !     0     0     0 !     0     0     0 !     0     0     0
   2 !     0     0     0 !     0     0     0 !     0     0     0
   3 !     0     0     0 !     0     0     0 !     0     0     0

          !  cst          !       code_cst      !       val_cst
   1       0              !     0     0     0 !     0     0     0
   2       0              !     0     0     0 !     0     0     0
   3       0              !     0     0     0 !     0     0     0

          !  variable     !       code_var      !       val_var
   1       1              !     0     0     0 !     0     0     0
   2       2              !     0     0     0 !     0     0     0
   3       3              !     0     0     0 !     0     0     0

mult :
   1    3.000
   2   -4.000
   3    5.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          !       val_ptt
     0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som   !     code1_som       !       val1_som
   1 !      0     0     0 !     0     0     0 !     0     0     0
   2 !      0     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst       !       val1_cst
   1 !      0              !     0     0     0 !     0     0     0
   2 !      0              !     0     0     0 !     0     0     0

             var1         !     code1_var       !       val1_var
   1 !      1              !     0     0     0 !     0     0     0
   2 !      2              !     0     0     0 !     0     0     0

     multi
   1  2.00000
   2  7.00000

2nd membre :

            no2_ent_som   !     code2_som       !       val2_som
   1 !      0     0     0 !     0     0     0 !     0     0     0

             cst2         !     code2_cst       !       val2_cst
   1 !      0              !     0     0     0 !     0     0     0
```

```
    mult2
  1  0.00000

Variables :
"x"   0,  0,  0,
"y"   0,  0,  0,
"z"   0,  0,  0,
   equation   2  nom : "deux"
      no_ent_ptt          code_ptt            val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som   !    code1_som      !    val1_som
  1  !     0     0     0 !     0     0     0 !     0     0     0
  2  !     0     0     0 !     0     0     0 !     0     0     0

             cst1         !    code1_cst      !    val1_cst
  1  !     0              !     0     0     0 !     0     0     0
  2  !     0              !     0     0     0 !     0     0     0

             var1         !    code1_var      !    val1_var
  1  !     2              !     0     0     0 !     0     0     0
  2  !     3              !     0     0     0 !     0     0     0

    mult1
  1  8.00000
  2  9.00000

2nd membre :

            no2_ent_som   !    code2_som      !    val2_som
  1  !     0     0     0 !     0     0     0 !     0     0     0

             cst2         !    code2_cst      !    val2_cst
  1  !     0              !     0     0     0 !     0     0     0

    mult2
  1 10.00000

Variables :
"x"   0,  0,  0,
"y"   0,  0,  0,
"z"   0,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   3.0 -4.0  5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

1	2.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
2	0.0	8.0	9.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		10.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

```
equ : min [3 + x - 4 * y + 5];
      un, 2 * x + 7 * y >= 6;
fin :
```

Fonction objectif : Il faut minimiser la fonction.

no_elem	no_ent_som			code_som			val_som		
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0

	cst	code_cst			val_cst		
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

	variable	code_var			val_var		
1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0
3	2	0	0	0	0	0	0
4	0	0	0	0	0	0	0

mult :

1	3.000
2	1.000
3	-4.000
4	5.000

Equations :
equation 1 nom : "un"

no_ent_ptt			code_ptt			val_ptt		
0	0	0	0	0	0	0	0	0

Code = 2

1er membre :

	no1_ent_som			code1_som			val1_som		
1	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0

```
            cst1          !      code1_cst      !     val1_cst
  1  !      0             !      0      0     0 !      0      0      0
  2  !      0             !      0      0     0 !      0      0      0

            var1          !      code1_var      !     val1_var
  1  !      1             !      0      0     0 !      0      0      0
  2  !      2             !      0      0     0 !      0      0      0

    mult1
  1  2.00000
  2  7.00000

2nd membre :

           no2_ent_som    !     code2_som       !     val2_som
  1  !      0      0      0 !      0      0     0 !      0      0      0

            cst2          !      code2_cst      !     val2_cst
  1  !      0             !      0      0     0 !      0      0      0

    mult2
  1  6.00000

Variables :
"x"    0,   0,   0,
"y"    0,   0,   0,

   Résultats de la transformation
Tableau "C" :

  ||    1.0 -4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  2.0  7.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   6.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -8.00
  w =  0.00


equ : min [3 + 5];
      un, 2 * x + 7 * y >= 6;
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som       !      val_som
  1 !    0      0     0 !      0      0     0 !      0      0      0
  2 !    0      0     0 !      0      0     0 !      0      0      0
```

```
        !  cst              !       code_cst        !       val_cst
  1      0                  !     0     0     0 !     0     0     0
  2      0                  !     0     0     0 !     0     0     0

        !  variable         !       code_var        !       val_var
  1      0                  !     0     0     0 !     0     0     0
  2      0                  !     0     0     0 !     0     0     0

mult :
  1    3.000
  2    5.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =    2

1er membre :

             no1_ent_som    !     code1_som       !     val1_som
  1  !     0     0     0 !     0     0     0 !     0     0     0
  2  !     0     0     0 !     0     0     0 !     0     0     0

              cst1          !     code1_cst       !     val1_cst
  1  !     0                !     0     0     0 !     0     0     0
  2  !     0                !     0     0     0 !     0     0     0

              var1          !     code1_var       !     val1_var
  1  !     1                !     0     0     0 !     0     0     0
  2  !     2                !     0     0     0 !     0     0     0

     mult1
  1  2.00000
  2  7.00000

2nd membre :

             no2_ent_som    !     code2_som       !     val2_som
  1  !     0     0     0 !     0     0     0 !     0     0     0

              cst2          !     code2_cst       !     val2_cst
  1  !     0                !     0     0     0 !     0     0     0

     mult2
  1  6.00000

Variables :
"x"   0,  0,  0,
"y"   0,  0,  0,

   Résultats de la transformation
Tableau "C" :

   ‖   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   ‖ ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

1	2.0	7.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		6.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

```
  z = -8.00
  w =  0.00
```

```
fam : i = [1_3];
      j = [1_2];
equ : min [3 * som(i,j) x(i,j) + 5 * som(i) y(i)];
      un, #j, 2 * som(i) x(i,j) + 7 * som(i) y(i) >= 6;
fin :
```

```
Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1     2     0 !     0     0     0 !     0     0     0
  2 !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst        !       val_cst
  1       0             !     0     0     0 !     0     0     0
  2       0             !     0     0     0 !     0     0     0

        !  variable     !       code_var        !       val_var
  1       1             !     0     0     0 !     0     0     0
  2       2             !     0     0     0 !     0     0     0
```

```
mult :
  1    3.000
  2    5.000
```

```
Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    2     0     0 !     0     0     0 !     0     0     0
Code =   2
```

```
1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0
  2  !      1     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst       !     val1_cst
  1  !      0             !     0     0     0 !     0     0     0
  2  !      0             !     0     0     0 !     0     0     0
```

```
            var1          !     code1_var      !     val1_var
  1  !      1             !     0     0     0  !     0     0     0
  2  !      2             !     0     0     0  !     0     0     0

    mult1
  1  2.00000
  2  7.00000

2nd membre :

            no2_ent_som   !     code2_som      !     val2_som
  1  !      0     0     0  !     0     0     0  !     0     0     0

             cst2         !     code2_cst      !     val2_cst
  1  !      0             !     0     0     0  !     0     0     0

    mult2
  1  6.00000

Variables :
"x"    1,   2,   0,
"y"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

  ||   3.0  3.0  3.0  3.0  3.0  3.0  5.0  5.0  5.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  2.0  0.0  2.0  0.0  2.0  0.0  7.0  7.0  7.0  0.0  ||   6.0
 2 ||  0.0  2.0  0.0  2.0  0.0  2.0  7.0  7.0  7.0  0.0  ||   6.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
equ : min [3 * som(i,j) x(i,j) + 5 * som(i,j) x(i,j)];
      un, #j, 2 * som(i) x(i,j) >= 6;
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som  !      code_som      !      val_som
  1 !    1     2     0 !     0     0     0 !     0     0     0
  2 !    1     2     0 !     0     0     0 !     0     0     0
```

```
        !  cst          !       code_cst        !       val_cst
                        !     0     0     0 !     0     0     0
1       0               !     0     0     0 !     0     0     0
2       0               !

        !  variable     !       code_var        !       val_var
                        !     0     0     0 !     0     0     0
1       1               !     0     0     0 !     0     0     0
2       1               !

mult :
  1     3.000
  2     5.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    2     0     0 !     0     0     0 !     0     0     0
Code =    2

1er membre :

             no1_ent_som    !     code1_som       !     val1_som
  1  !       1     0     0 !     0     0     0 !     0     0     0

              cst1          !     code1_cst       !     val1_cst
  1  !       0              !     0     0     0 !     0     0     0

              var1          !     code1_var       !     val1_var
  1  !       1              !     0     0     0 !     0     0     0

     mult1
  1  2.00000

2nd membre :

             no2_ent_som    !     code2_som       !     val2_som
  1  !       0     0     0 !     0     0     0 !     0     0     0

              cst2          !     code2_cst       !     val2_cst
  1  !       0              !     0     0     0 !     0     0     0

     mult2
  1  6.00000

Variables :
"x"    1,   2,   0,

   Résultats de la transformation
Tableau "C" :

   ||    8.0  8.0  8.0  8.0  8.0  8.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

1	2.0	0.0	2.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0		6.0
2	0.0	2.0	0.0	2.0	0.0	2.0	0.0	0.0	0.0	0.0		6.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
      j = [1_2];
equ : min [3 * som(i) x(i) + 5 * som(j) y(j)];
      un, som(i) 2 * x(i) + som(j) 4 * y(j) >= 6;
fin :
```

Fonction objectif : Il faut minimiser la fonction.

no_elem	no_ent_som			code_som			val_som		
1	1	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0

	cst	code_cst			val_cst		
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0

	variable	code_var			val_var		
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0

mult :

1	3.000
2	5.000

Equations :

equation 1 nom : "un"

no_ent_ptt			code_ptt			val_ptt		
0	0	0	0	0	0	0	0	0

Code = 2

1er membre :

	no1_ent_som			code1_som			val1_som		
1	1	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0

	cst1	code1_cst			val1_cst		
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0

```
                var1                !      code1_var          !      val1_var
1   !      1                        !      0      0      0    !      0      0      0
2   !      2                        !      0      0      0    !      0      0      0

   mult1
1   2.00000
2   4.00000

2nd membre :

               no2_ent_som    !      code2_som          !      val2_som
1   !      0      0      0    !      0      0      0    !      0      0      0

                cst2          !      code2_cst          !      val2_cst
1   !      0                  !      0      0      0    !      0      0      0

   mult2
1   6.00000

Variables :
"x"    1,   0,   0,
"y"    2,   0,   0,

   Résultats de la transformation
Tableau "C" :

   ||   3.0  3.0  3.0  5.0  5.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  2.0  2.0  2.0  4.0  4.0  0.0  0.0  0.0  0.0  0.0  ||  6.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

   z =  0.00
   w =  0.00


fam : i = [1_3];
      j = [1_2];
equ : min [3 * som(i) x(i) + 5 * som(j) y(j)];
      un, 8 * som(i) 2 * x(i) + 7 * som(j) 4 * y(j) >= 6;
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som           !      val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0
  2 !     2      0      0 !      0      0      0 !      0      0      0
```

```
        !  cst          !       code_cst       !      val_cst
  1      0              !     0     0     0 !      0     0     0
  2      0              !     0     0     0 !      0     0     0

        !  variable     !       code_var       !      val_var
  1      1              !     0     0     0 !      0     0     0
  2      2              !     0     0     0 !      0     0     0

mult :
  1    3.000
  2    5.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt           val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      1     0     0 !      0     0     0 !      0     0     0
  2  !      2     0     0 !      0     0     0 !      0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !      0              !     0     0     0 !      0     0     0
  2  !      0              !     0     0     0 !      0     0     0

             var1          !     code1_var       !     val1_var
  1  !      1              !     0     0     0 !      0     0     0
  2  !      2              !     0     0     0 !      0     0     0

    mult1
  1 16.00000
  2 28.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2          !     code2_cst       !     val2_cst
  1  !      0              !     0     0     0 !      0     0     0

    mult2
  1  6.00000

Variables :
"x"   1,  0,  0,
"y"   2,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   3.0  3.0  3.0  5.0  5.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

1	16.0	16.0	16.0	28.0	28.0	0.0	0.0	0.0	0.0	0.0		6.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i) c(i) * x(i)];
      un, 8 * som(i) c(i) * x(i) >= 6;
fin :
```

Fonction objectif : Il faut minimiser la fonction.

no_elem	no_ent_som			code_som			val_som		
1	1	0	0	0	0	0	0	0	0

	cst	code_cst			val_cst		
1	1	0	0	0	0	0	0

	variable	code_var			val_var		
1	1	0	0	0	0	0	0

mult :
1 1.000

Equations :
equation 1 nom : "un"

no_ent_ptt			code_ptt			val_ptt		
0	0	0	0	0	0	0	0	0

Code = 2

1er membre :

	no1_ent_som			code1_som			val1_som		
1	1	0	0	0	0	0	0	0	0

	cst1	code1_cst			val1_cst		
1	1	0	0	0	0	0	0

	var1	code1_var			val1_var		
1	1	0	0	0	0	0	0

	mult1
1	8.00000

```
2nd membre :

          no2_ent_som   !     code2_som       !     val2_som
  1  !     0     0     0 !      0     0     0 !      0     0     0

           cst2         !     code2_cst       !     val2_cst
  1  !     0            !      0     0     0 !      0     0     0

    mult2
  1  6.00000

Variables :
"x"   1,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   1.0  2.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  8.0 16.0 24.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   6.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i) c(i) * x(i)];
      un, 8 * som(i) 4 * c(i) * x(i) >= 6;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !      code_som       !      val_som
  1  !     1     0     0 !      0     0     0 !      0     0     0

        !  cst          !      code_cst       !      val_cst
  1      1              !      0     0     0 !      0     0     0

        !  variable     !      code_var       !      val_var
  1      1              !      0     0     0 !      0     0     0

mult :
  1     1.000
```

```
Equations :
   equation    1   nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =    2

1er membre :

            no1_ent_som     !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst1           !     code1_cst       !     val1_cst
  1  !      1              !     0     0     0 !     0     0     0

             var1           !     code1_var       !     val1_var
  1  !      1              !     0     0     0 !     0     0     0

    mult1
  1 32.00000

2nd membre :

            no2_ent_som     !     code2_som       !     val2_som
  1  !      0     0     0 !     0     0     0 !     0     0     0

             cst2           !     code2_cst       !     val2_cst
  1  !      0              !     0     0     0 !     0     0     0

    mult2
  1  6.00000

Variables :
"x"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

   ||   1.0  2.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

  1 || 32.0 64.0 96.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  6.0
  2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
  9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0

   z =  0.00
   w =  0.00
```

```
fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(1) + 15 * som(j) k(j) * y(j)];
      un, #j, 8 * som(i) c(i) * x(i) + k(j) * y(j) >= 6;
fin :
```

Fonction objectif : Il faut minimiser la fonction.

no_elem	no_ent_som			code_som			val_som		
1	1	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0

	cst	code_cst			val_cst		
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0

	variable	code_var			val_var		
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0

mult :

1	1.000
2	15.000

Equations :

equation 1 nom : "un"

no_ent_ptt			code_ptt			val_ptt		
2	0	0	0	0	0	0	0	0

Code = 2

1er membre :

	no1_ent_som			code1_som			val1_som		
1	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0

	cst1	code1_cst			val1_cst		
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0

	var1	code1_var			val1_var		
1	1	0	0	0	0	0	0
2	2	0	0	0	0	0	0

	mult1
1	8.00000
2	1.00000

2nd membre :

	no2_ent_som			code2_som			val2_som		
1	0	0	0	0	0	0	0	0	0

	cst2	code2_cst			val2_cst		
1	0	0	0	0	0	0	0

	mult2
1	6.00000

```
Variables :
"x"    1,   0,   0,
"y"    2,   0,   0,

   Résultats de la transformation
Tableau "C" :

   ||    1.0   2.0   3.0 150.0 300.0   0.0   0.0   0.0   0.0   0.0
Tableau "D" :

   || !  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0
Tableaux "A" et "B"

 1 ||   8.0  16.0  24.0  10.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   6.0
 2 ||   8.0  16.0  24.0   0.0  20.0   0.0   0.0   0.0   0.0   0.0  ||   6.0
 3 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 4 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 5 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 6 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 7 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 8 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
 9 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0
10 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||   0.0

  z =   0.00
  w =   0.00


fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i<>1) c(i) * x(i) + 15 * som(j<>2) k(j) * y(j)];
      un, #j<>1, 8 * som(i<>3) c(i) * x(i) + k(j) * y(j) >= 6;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1       0     0 !     1       0     0 !     1       0     0
  2 !     2       0     0 !     1       0     0 !     2       0     0

        !  cst          !       code_cst        !       val_cst
  1      1              !     0       0     0 !     0       0     0
  2      2              !     0       0     0 !     0       0     0

        !  variable     !       code_var        !       val_var
  1      1              !     0       0     0 !     0       0     0
  2      2              !     0       0     0 !     0       0     0

mult :
  1     1.000
  2    15.000
```

```
Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    2     0     0 !     1     0     0 !     1     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som      !     val1_som
  1  !      1     0     0 !      1     0     0 !      3     0     0
  2  !      0     0     0 !      0     0     0 !      0     0     0

             cst1          !     code1_cst      !     val1_cst
  1  !      1              !      0     0     0 !      0     0     0
  2  !      2              !      0     0     0 !      0     0     0

             var1          !     code1_var      !     val1_var
  1  !      1              !      0     0     0 !      0     0     0
  2  !      2              !      0     0     0 !      0     0     0

    mult1
  1  8.00000
  2  1.00000

2nd membre :

            no2_ent_som    !     code2_som      !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2          !     code2_cst      !     val2_cst
  1  !      0              !      0     0     0 !      0     0     0

    mult2
  1  6.00000

Variables :
"x"    1,  0,  0,
"y"    2,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   0.0  2.0  3.0 150.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  8.0 16.0  0.0  0.0 20.0  0.0  0.0  0.0  0.0  0.0 ||  6.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
```

```
  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [c(i)<1) * x(i)<1) + 15 * som(j) k(j)<2) * y(j)];
      un, som(i) 8 * c(i) * x(i) + som (j) k(j) * y(j)<2) >= 6;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     0     0     0 !     0     0     0 !     0     0     0
  2 !     2     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst      !       val_cst
  1       1             !     1     0     0 !     1     0     0
  2       2             !     1     0     0 !     2     0     0

        !  variable     !       code_var      !       val_var
  1       1             !     1     0     0 !     1     0     0
  2       2             !     0     0     0 !     0     0     0

mult :
  1     1.000
  2    15.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =    2

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1 !       1     0     0 !     0     0     0 !     0     0     0
  2 !       2     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst       !     val1_cst
  1 !       1             !     0     0     0 !     0     0     0
  2 !       2             !     0     0     0 !     0     0     0

             var1         !     code1_var       !     val1_var
  1 !       1             !     0     0     0 !     0     0     0
  2 !       2             !     1     0     0 !     2     0     0

    mult1
  1  8.00000
  2  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1 !       0     0     0 !     0     0     0 !     0     0     0
```

```
          cst2          !     code2_cst      !     val2_cst
  1  !     0             !     0     0     0 !     0     0     0

    mult2
  1  6.00000

Variables :
"x"    1,  0,  0,
"y"    2,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ‖   1.0  0.0  0.0 300.0 300.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  ‖ ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ‖  8.0 16.0 24.0  0.0 30.0  0.0  0.0  0.0  0.0  0.0 ‖   6.0
 2 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 3 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 4 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 5 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 6 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 7 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 8 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
 9 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0
10 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) k(j)<2) * y(j)];
      un, som(i) 8 * c(i)<ibis) * x(i) + som (j) k(j) * y(j)<2) >= 6;
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som      !      val_som
  1 !    1     0     0 !     0     0     0 !     0     0     0
  2 !    3     0     0 !     0     0     0 !     0     0     0

        !  cst          !      code_cst      !      val_cst
  1      1              !     0     0     0 !     0     0     0
  2      2              !     1     0     0 !     2     0     0

        !  variable     !      code_var      !      val_var
  1      1              !     3     0     0 !     2     0     0
  2      2              !     0     0     0 !     0     0     0

mult :
  1    1.000
  2   15.000
```

```
Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
   0     0     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som      !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0
  2  !      3     0     0 !     0     0     0 !     0     0     0

             cst1          !     code1_cst      !     val1_cst
  1  !      1              !     3     0     0 !     2     0     0
  2  !      2              !     0     0     0 !     0     0     0

             var1          !     code1_var      !     val1_var
                           !     0     0     0 !     0     0     0
  1  !      1              !     1     0     0 !     2     0     0
  2  !      2              !

    mult1
  1  8.00000
  2  1.00000

2nd membre :

            no2_ent_som    !     code2_som      !     val2_som
  1  !      0     0     0 !     0     0     0 !     0     0     0

             cst2          !     code2_cst      !     val2_cst
  1  !      0              !     0     0     0 !     0     0     0

    mult2
  1  6.00000

Variables :
"x"    1,   0,   0,
"y"    3,   0,   0,
```

**** erreur (70) du type indice resultant hors bornes ****

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(j) * x(i)<ibis) + 15 * som(j) k(j><2) * y(j)];
```

**** erreur (26) du type déclaration incohérente ****

```
      k(j) = 10,20;
equ : min [som(i) c(j) * x(i)<ibis) + 15 * som(j) k(j><2) * y(j)];
                                  ^
```

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) k(j)<2) * y(j)];
      un, som(i) 8 * c(i)<ibis) * x(j) + som (j) k(j) * y(j)<2) >= 6;
fin :
```

**** erreur (26) du type déclaration incohérente ****

```
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) k(j)<2) * y(j)];
      un, som(i) 8 * c(i)<ibis) * x(j) + som (j) k(j) * y(j)<2) >= 6;
                                         ^
```

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(i)];
```

**** erreur (26) du type déclaration incohérente ****

```
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(i)];
                                                    ^
```

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];
      un, som(i) 8 * c(i)<ibis) * x(i) + k(j) * y(j)<2) >= 6;
fin :
```

**** erreur (65) du type famille superflue ****

```
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];
      un, som(i) 8 * c(i)<ibis) * x(i) + k(j) * y(j)<2) >= 6;
                                                       ^
```

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<j) + 15 * som(j) y(j)];
```

**** erreur (54) du type décalage impossible ****

```
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<j) + 15 * som(j) y(j)];
                          ^
```

```
fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];
      un, #i, som(i) 8 * c(i)<ibis) * x(i) + som(j) k(j) * y(j)<2) >= 6;

fin :

*** erreur (64) du type famille apparaissant dans "#" et dans "som" **


equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];
      un, #i, som(i) 8 * c(i)<ibis) * x(i) + som(j) k(j) * y(j)<2) >= 6;
                                                                  ^


fam : i = [1_3];
      ibis = (wavre,bxl,namur);
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];

       *** erreur (54) du type décalage impossible ***

      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<ibis) + 15 * som(j) y(j)];
                                    ^


fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<4) + 15 * som(j) y(j)];

       *** erreur (39) du type valeur non comprise dans les bornes ***

      k(j) = 10,20;
equ : min [som(i) c(i) * x(i)<4) + 15 * som(j) y(j)];
                               ^


fam : i = [1_3];
      ibis = [3_5];
      j = [1_2];
cst : c(i) = 1,2,3;
      k(j) = 10,20;
equ : min [som(i) c(i)<4) * x(i) + 15 * som(j) y(j)];

       *** erreur (39) du type valeur non comprise dans les bornes ***

      k(j) = 10,20;
equ : min [som(i) c(i)<4) * x(i) + 15 * som(j) y(j)];
                          ^
```

```
equ : min[2*x + 4*t - 3*y - 4*z + 5*t - 6*x + 7*z -8*y];
```

*** erreur (46) du type trop d'éléments de somme ***

```
****** debut du fichier *****
equ : min[2*x + 4*t - 3*y - 4*z + 5*t - 6*x + 7*z -8*y];
                                              ^
```

```
equ : min[2*x + 3*y - 4*z];
      un, 2*z - 3*x + 4*y - 5*z - 6*y + 7*x - 4*t = - 8;
fin :
```

*** erreur (46) du type trop d'éléments de somme ***

```
      un, 2*z - 3*x + 4*y - 5*z - 6*y + 7*x - 4*t = - 8;
fin :
    ^
```

```
equ : min[2*x + 3*y - 4*z];
      un, 2*z - 3*x + 4*y - 5*z -6*y + 7*x - 8*z + 5*t >= 1;
```

*** erreur (46) du type trop d'éléments de somme ***

```
equ : min[2*x + 3*y - 4*z];
      un, 2*z - 3*x + 4*y - 5*z -6*y + 7*x - 8*z + 5*t >= 1;
                                               ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i,j) c(i)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i) = 1,2,3;
equ : min [som(i,j) c(i)];
                    ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i,j) x(i)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i) = 1,2,3;
equ : min [som(i,j) x(i)];
                    ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i,j) c(i) * x(i)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i) = 1,2,3;
equ : min [som(i,j) c(i) * x(i)];
                           ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j><1)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j><1)];
                           ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i,j) x(i,j><1)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i) = 1,2,3;
equ : min [som(i,j) x(i,j><1)];
                           ^
```

```
fam : i =[1_3];
      j = [1_2];
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j><1) * x(i,j><1)];
fin :
```

*** erreur (65) du type famille superflue ***

```
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j><1) * x(i,j><1)];
                                        ^
```

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j) * x(i,j)];
      un, #j, som(i,k) x(i,j) = som(i) c(i,j)
fin :
```

*** erreur (65) du type famille superflue ***

```
equ : min [som(i,j) c(i,j) * x(i,j)];
      un, #j, som(i,k) x(i,j) = som(i) c(i,j)
                                ^
```

```
equ : min[x + y];
      un, x + 3 <= 0;
fin :
```

*** erreur (63) du type réel hors position ***

```
equ : min[x + y];
      un, x + 3 <= 0;
                ^
```

```
equ : min[x + y];
      un, x <= y;
fin :
```

*** erreur (50) du type variable hors position ***

```
equ : min[x + y];
      un, x <= y;
                ^
```

```
cst : c = 10;
equ : min[x + y];
      un, x <= c * y;
fin :
```

*** erreur (50) du type variable hors position ***

```
equ : min[x + y];
      un, x <= c * y;
                  ^
```

```
cst : c = 10;
equ : min[x + y];
      un, x + c <= 2;
fin :
```

*** erreur (62) du type constante hors position ***

```
equ : min[x + y];
      un, x + c <= 2;
                ^
```

```
cst : c = 10;
equ : min[x + y];
      un, x + 3 * c <= 2;
fin :
```

*** erreur (62) du type constante hors position ***

```
equ : min[x + y];
      un, x + 3 * c <= 2;
                    ^
```

```
fam : ville = (bxl,namur,liege);
equ : max[som(ville <> wavre) x(ville)];
fin :
```

*** erreur (41) du type nom non repris dans l'énumération ***

```
fam : ville = (bxl,namur,liege);
equ : max[som(ville <> wavre) x(ville)];
                    ^
```

```
fam : ville = (bxl,namur,liege);
equ : max[som(ville) x(ville >< wavre)];
fin :
```

*** erreur (41) du type nom non repris dans l'énumération ***

```
fam : ville = (bxl,namur,liege);
equ : max[som(ville) x(ville >< wavre)];
                              ^
```

```
fam : ville = (bxl,namur,liege);
cst : c(ville) = 1,2,3;
equ : max[som(ville) c(ville >< wavre)];
fin :
```

*** erreur (41) du type nom non repris dans l'énumération ***

```
cst : c(ville) = 1,2,3;
equ : max[som(ville) c(ville >< wavre)];
                              ^
```

```
fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i,i) = 1,2,3,4,5,6,7,8,9;
equ : min [som(i) c2(i >< 1,i)];
      un, #j, som(i) x(i) <= som(i) c1(i,j);
fin :
```

```
Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som       !       val_som
   1 !     1     0     0 !     0     0     0 !     0     0     0
```

```
        !  cst             !       code_cst       !       val_cst
  1      2                 !     1      0      0 !      1      0      0

        !  variable        !       code_var       !       val_var
  1      0                 !     0      0      0 !      0      0      0

mult :
  1    1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    2      0      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

             no1_ent_som    !     code1_som       !     val1_som
  1  !       1      0      0 !      0      0      0 !      0      0      0

              cst1          !     code1_cst       !     val1_cst
  1  !       0              !     0      0      0 !      0      0      0

              var1          !     code1_var       !     val1_var
  1  !       1              !     0      0      0 !      0      0      0

     mult1
  1  1.00000

2nd membre :

             no2_ent_som    !     code2_som       !     val2_som
  1  !       1      0      0 !      0      0      0 !      0      0      0

              cst2          !     code2_cst       !     val2_cst
  1  !       1              !     0      0      0 !      0      0      0

     mult2
  1  1.00000

Variables :
"x"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

```
Tableaux "A" et "B"

 1 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   9.0
 2 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -3.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 10,20,30;
equ : min [som(i) x(i >< 1,i)];
      un, som(i) x(i,i) <= som(i) c2(i);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst      !       val_cst
  1      0              !     0     0     0 !     0     0     0

        !  variable     !       code_var      !       val_var
  1      1              !     1     0     0 !     1     0     0

mult :
  1    1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          !       val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst1         !      code1_cst      !     val1_cst
  1  !      0              !     0     0     0 !     0     0     0

             var1         !      code1_var      !     val1_var
  1  !      1              !     0     0     0 !     0     0     0

    mult1
  1  1.00000
```

```
2nd membre :

          no2_ent_som   !    code2_som      !    val2_som
  1  !     1     0     0 !     0     0     0 !     0     0     0

           cst2         !    code2_cst      !    val2_cst
  1  !     2            !     0     0     0 !     0     0     0

    mult2
  1  1.00000

Variables :
"x"   1,  1,  0,

   Résultats de la transformation
Tableau "C" :

  ‖   1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  ‖ ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ‖  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ‖  60.0
 2 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 3 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 4 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 5 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 6 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 7 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 8 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
 9 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0
10 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ‖   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 10,20,30;
equ : min [som(i) c2(i)<1) * x(i >< 1,i)];
      un, som(i) x(i,i) <= som(i) c2(i);
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som      !      val_som
  1  !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !      code_cst      !      val_cst
  1      2              !     1     0     0 !     1     0     0

        !  variable     !      code_var      !      val_var
  1      1              !     1     0     0 !     1     0     0

mult :
  1    1.000
```

```
Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !     0     0     0 !     0     0     0

             var1          !     code1_var       !     val1_var
  1  !      1             !     0     0     0 !     0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst2          !     code2_cst       !     val2_cst
  1  !      2             !     0     0     0 !     0     0     0

    mult2
  1  1.00000

Variables :
"x"   1,  1,  0,

   Résultats de la transformation
Tableau "C" :

  ||  10.0 10.0 10.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ||  60.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

   z =  0.00
   w =  0.00
```

```
fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i,i) = 1,2,3,4,5,6,7,8,9;
equ : min [som(i) c2(i><1,i) * x(i >< 1)];
      un, #j, som(i) x(i) <= som(i) c1(i,j);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst      !       val_cst
  1      2              !     1     0     0 !     1     0     0

        !  variable     !       code_var      !       val_var
  1      1              !     1     0     0 !     1     0     0

mult :
  1    1.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    2     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst       !     val1_cst
  1  !      0             !     0     0     0 !     0     0     0

             var1         !     code1_var       !     val1_var
  1  !      1             !     0     0     0 !     0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst2         !     code2_cst       !     val2_cst
  1  !      1             !     0     0     0 !     0     0     0

    mult2
  1  1.00000

Variables :
"x"   1,  0,  0,

   Résultats de la transformation
Tableau "C" :

  ||   6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableau "D" :

  ‖ ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ‖  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  9.0
 2 ‖  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖ 12.0
 3 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 4 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 5 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 6 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 7 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 8 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
 9 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0
10 ‖  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ‖  0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i,i) = 1,2,3,4,5,6,7,8,9;
equ : min [som(i) c2(i,i)<1) * x(i )< 1)];
      un, #j, som(i) x(i) <= som(i) c1(i,j);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !      code_som      !      val_som
  1 !     1     0     0 !     0     0     0 !     0     0     0

        !  cst          !      code_cst      !      val_cst
  1      2              !     0     1     0 !     0     1     0

        !  variable     !      code_var      !      val_var
  1      1              !     1     0     0 !     1     0     0

mult :
  1    1.000


Equations :
   equation   1  nom : "un"
     no_ent_ptt          code_ptt          val_ptt
    2     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

           no1_ent_som   !    code1_som      !    val1_som
  1  !     1     0     0 !     0     0     0 !     0     0     0

            cst1         !     code1_cst     !     val1_cst
  1  !     0              !     0     0     0 !     0     0     0

            var1         !     code1_var     !     val1_var
  1  !     1              !     0     0     0 !     0     0     0
```

```
    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

             cst2          !     code2_cst       !     val2_cst
  1  !      1             !      0      0      0 !      0      0      0

    mult2
  1  1.00000

Variables :
"x"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

  ||  12.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   9.0
 2 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 1,2,3;
equ : min [som(i) c2(i><1) * x(i,i >< 1)];
      un, #j, som(i) x(i,i) <= som(i) c1(i,j);
fin :

Fonction objectif :    Il faut minimiser la fonction.
no_elem ! no_ent_som   !      code_som       !      val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0

        !  cst          !      code_cst       !      val_cst
  1      2              !      1      0      0 !      1      0      0

        !  variable     !      code_var       !      val_var
  1      1              !      0      1      0 !      0      1      0
```

```
mult :
  1     1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    2      0      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

             no1_ent_som    !     code1_som       !     val1_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

              cst1          !     code1_cst       !     val1_cst
  1  !      0              !      0      0      0 !      0      0      0

              var1          !     code1_var       !     val1_var
  1  !      1              !      0      0      0 !      0      0      0

    mult1
  1  1.00000

2nd membre :

             no2_ent_som    !     code2_som       !     val2_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

              cst2          !     code2_cst       !     val2_cst
  1  !      1              !      0      0      0 !      0      0      0

    mult2
  1  1.00000

Variables :
"x"    1,   1,   0,

   Résultats de la transformation
Tableau "C" :

   ||    1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ||   9.0
 2 ||  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ||  12.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

   z =  0.00
   w =  0.00
```

```
fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 1,2,3;
equ : min [som(i) x(i,i >< 1)];
      un, #j, som(i) x(i,i) <= som(i) c1(i,j);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0

        !  cst          !       code_cst      !       val_cst
  1      0              !      0      0      0 !      0      0      0

        !  variable     !       code_var      !       val_var
  1      1              !      0      1      0 !      0      1      0

mult :
  1     1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    2      0      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

             cst1         !      code1_cst      !      val1_cst
  1  !      0              !      0      0      0 !      0      0      0

             var1         !      code1_var      !      val1_var
  1  !      1              !      0      0      0 !      0      0      0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

             cst2         !      code2_cst      !      val2_cst
  1  !      1              !      0      0      0 !      0      0      0

    mult2
  1  1.00000

Variables :
"x"    1,   1,   0,
```

```
   Résultats de la transformation
Tableau "C" :

  ||   1.0  0.0  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ||   9.0
 2 ||  1.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  1.0  0.0  ||  12.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i,i) = 1,2,3,4,5,6,7,8,9;
equ : min [som(i) c2(i,i >< 1)];
      un, #j, som(i) x(i,i) <= som(i) c1(i,j);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1      0      0 !      0      0      0 !      0      0      0

        !  cst          !       code_cst      !       val_cst
  1      2              !      0      1      0 !      0      1      0

        !  variable     !       code_var      !       val_var
  1      0              !      0      0      0 !      0      0      0

mult :
  1     1.000


Equations :
   equation  1  nom : "un"
      no_ent_ptt          code_ptt          !          val_ptt
    2      0      0 !      0      0      0 !      0      0      0
Code =   3

1er membre :

            no1_ent_som   !     code1_som      !     val1_som
  1  !      1      0      0 !      0      0      0 !      0      0      0

             cst1         !      code1_cst     !      val1_cst
  1  !      0              !      0      0      0 !      0      0      0
```

```
                var1                !       code1_var       !       val1_var
  1  !       1                      !       0       0       0 !       0       0       0

    mult1
  1  1.00000

2nd membre :

               no2_ent_som    !       code2_som       !       val2_som
  1  !       1       0       0 !       0       0       0 !       0       0       0

                cst2                !       code2_cst       !       val2_cst
  1  !       1                      !       0       0       0 !       0       0       0

    mult2
  1  1.00000

Variables :
"x"    1,   1,   0,

   Résultats de la transformation
Tableau "C" :

   ||    0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||   1.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0   0.0   ||    9.0
 2 ||   1.0   0.0   0.0   0.0   1.0   0.0   0.0   0.0   1.0   0.0   ||   12.0
 3 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 4 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 5 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 6 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 7 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 8 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 9 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
10 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0

  z = -3.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c(i) = 1,2,3;
equ : min [som(i) c(i) * x(i >< 1)];
      un, som(i) c(i >< 1) * x(i) <= 2;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1       0       0 !       0       0       0 !       0       0       0

        !  cst          !       code_cst        !       val_cst
  1       1             !       0       0       0 !       0       0       0
```

```
        !  variable    !       code_var       !      val_var
  1      1              !     1     0     0 !      1     0     0

mult :
  1    1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    0     0     0 !      0     0     0 !      0     0     0
Code =    3

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      1     0     0 !      0     0     0 !      0     0     0

             cst1          !      code1_cst      !     val1_cst
  1  !      1              !      1     0     0 !      1     0     0

             var1          !      code1_var      !     val1_var
  1  !      1              !      0     0     0 !      0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2          !      code2_cst      !     val2_cst
  1  !      0              !      0     0     0 !      0     0     0

    mult2
  1  2.00000

Variables :
"x"    1,   0,   0,

   Résultats de la transformation
Tableau "C" :

   ||   6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :

   || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  2.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
```

```
  z =  0.00
  w =  0.00


fam : i = [1_3];
      j = [1_2];
cst : c(i,j) = 1,2,3,4,5,6;
equ : min [som(i,j) c(i,j) * x(i,j)];
      un, #j, som(i) x(i,j >< 1) <= som(i) c(i,j >< 1);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     1     2     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst        !       val_cst
  1      1              !     0     0     0 !     0     0     0

        !  variable     !       code_var        !       val_var
  1      1              !     0     0     0 !     0     0     0

mult :
  1    1.000


Equations :
   equation   1  nom : "un"
     no_ent_ptt          code_ptt          val_ptt
    2     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst       !     val1_cst
  1  !      0             !     0     0     0 !     0     0     0

             var1         !     code1_var       !     val1_var
  1  !      1             !     0     1     0 !     0     1     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      1     0     0 !     0     0     0 !     0     0     0

             cst2         !     code2_cst       !     val2_cst
  1  !      1             !     0     1     0 !     0     1     0

    mult2
  1  1.00000

Variables :
"x"   1,  2,  0,
```

```
   Résultats de la transformation
Tableau "C" :

  ||   1.0  2.0  3.0  4.0  5.0  6.0  0.0  0.0  0.0  0.0
Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ||  9.0
 2 ||  1.0  0.0  1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ||  9.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

  z =  0.00
  w =  0.00


equ : max[3 * x + 5 * y];
com : blablabla
fin :

       *** erreur (22) du type "," attendu ***

equ : max[3 * x + 5 * y];
com : blablabla
    ^


equ : max[3 * x + 5 * y];
fin :

       *** erreur (22) du type "," attendu ***

equ : max[3 * x + 5 * y];
fin :
    ^


equ : 3 * x + 5 * y <= 4;

       *** erreur (38) du type "[" attendu ***

****** debut du fichier *****
equ : 3 * x + 5 * y <= 4;
        ^
```

```
equ : max[3 * x + 5 * y];
      123, x - 2 * y >= 6;

       *** erreur ( 1) du type nom invalide ***

equ : max[3 * x + 5 * y];
      123, x - 2 * y >= 6;
           ^


equ : min[x + y + z + t + k + l + m];
      un, n + o + p + q >= 2;

       *** erreur (67) du type fin de fichier inattendue ***

      un, n + o + p + q >= 2;

^


equ : max[2 * x + 3 * y];
      un, x + z <= 3;
      deux, y - t >= 2;
      trois, t + z <= 4;
      quatre, x + t <= 5;
      cinq, - y + x <= 1;
      six, x - y - z <= 1;

       *** erreur (67) du type fin de fichier inattendue ***

      six, x - y - z <= 1;

^


fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      l = [1_5];
equ : min [3 * x + 4 * y];
      un, #i, #j, #k, #l, x(i,j,k,l)<= 2;

       *** erreur (21) du type trop d'indices ***

equ : min [3 * x + 4 * y];
      un, #i, #j, #k, #l, x(i,j,k,l)<= 2;
                          ^
```

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      l = [1_5];
equ : min [som(i,j,k,l) x(i,j,k,l)];
```

*** erreur (21) du type trop d'indices ***

```
      l = [1_5];
equ : min [som(i,j,k,l) x(i,j,k,l)];
                        ^
```

```
fam : i = [1_3];
      j = [1_2];
      k = [1_4];
      l = [1_5];
equ : min [som(i,j,k) x(i,j,k)];
      un, y(i,j,k,l) <= 4;
```

*** erreur (21) du type trop d'indices ***

```
equ : min [som(i,j,k) x(i,j,k)];
      un, y(i,j,k,l) <= 4;
                     ^
```

```
fam : i = [1_3];
equ : min[ 3 * x + 4 * y];
      #i, x(i) <= 3;
```

*** erreur (1) du type nom invalide ***

```
equ : min[ 3 * x + 4 * y];
      #i, x(i) <= 3;
        ^
```

```
fam : i = [1_3];
equ : min[3 * x + 4 * y];
      som(i) x(i) <= 2;
```

*** erreur (22) du type "," attendu ***

```
equ : min[3 * x + 4 * y];
      som(i) x(i) <= 2;
          ^
```

```
fam : i = [1_3];
cst : c = 2;
equ : min[som(i) c * x(i)];
fin :

       *** erreur (22) du type "," attendu ***

equ : min[som(i) c * x(i)];
fin :
    ^



cst : c = 2;
equ : min [3 * 4 * c * x];
fin :

       *** erreur (22) du type "," attendu ***

equ : min [3 * 4 * c * x];
fin :
    ^



equ : min [3 * x + 4 * y];
      un, 3 * x - 5 * y <= 2 + 0;
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som       !       val_som
  1 !     0     0     0 !     0     0     0 !     0     0     0
  2 !     0     0     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst       !       val_cst
  1      0              !     0     0     0 !     0     0     0
  2      0              !     0     0     0 !     0     0     0

        !  variable     !       code_var       !       val_var
  1      1              !     0     0     0 !     0     0     0
  2      2              !     0     0     0 !     0     0     0

mult :
  1    3.000
  2    4.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    0     0     0 !     0     0     0 !     0     0     0
Code =   3

1er membre :

           no1_ent_som    !     code1_som       !     val1_som
  1  !     0     0     0 !     0     0     0 !     0     0     0
  2  !     0     0     0 !     0     0     0 !     0     0     0
```

```
                cst1                !       code1_cst          !      val1_cst
  1   !         0                   !       0       0       0  !       0       0       0
  2   !         0                   !       0       0       0  !       0       0       0

                var1                !       code1_var          !      val1_var
  1   !         1                   !       0       0       0  !       0       0       0
  2   !         2                   !       0       0       0  !       0       0       0

     mult1
  1   3.00000
  2  -5.00000

2nd membre :

                no2_ent_som         !       code2_som          !      val2_som
  1   !         0       0       0  !       0       0       0  !       0       0       0
  2   !         0       0       0  !       0       0       0  !       0       0       0

                cst2                !       code2_cst          !      val2_cst
  1   !         0                   !       0       0       0  !       0       0       0
  2   !         0                   !       0       0       0  !       0       0       0

     mult2
  1   2.00000
  2   0.00000

Variables :
"x"    0,   0,   0,
"y"    0,   0,   0,
   Résultats de la transformation

Tableau "C" :

   ||    3.0   4.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

Tableau "D" :

   ||  !  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0!  0.0
Tableaux "A" et "B"

  1 ||   3.0  -5.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    2.0
  2 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  3 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  4 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  5 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  6 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  7 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  8 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
  9 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0
 10 ||   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   ||    0.0

   z =   0.00
   w =   0.00
```

```
fam : i = [1_3];
      ibis = [0_2];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 10,20,30;
equ : min[som(i <> 1) c2(i >< ibis) * x(i >< ibis)];
      un, #i <> 1, som(j) y(i,j) = c2(i >< ibis);
fin :
Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !      code_som      !      val_som
  1 !    1     0     0 !     1     0     0 !     1     0     0

        ! cst            !      code_cst      !      val_cst
  1      2               !     3     0     0 !     2     0     0

        ! variable       !      code_var      !      val_var
  1      1               !     3     0     0 !     2     0     0

mult :
  1    1.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    1     0     0 !     1     0     0 !     1     0     0
Code =   1

1er membre :

            no1_ent_som   !    code1_som      !    val1_som
  1  !      3     0     0 !     0     0     0 !     0     0     0

             cst1         !     code1_cst     !    val1_cst
  1  !      0              !     0     0     0 !     0     0     0

             var1         !     code1_var     !    val1_var
  1  !      2              !     0     0     0 !     0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !    code2_som      !    val2_som
  1  !      0     0     0 !     0     0     0 !     0     0     0

             cst2         !     code2_cst     !    val2_cst
  1  !      2              !     3     0     0 !     2     0     0

    mult2
  1  1.00000

Variables :
"x"    1,  0,  0,
"y"    1,  3,  0,
```

Résultats de la transformation

Tableau "C" :

```
  ||  10.0 20.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableau "D" :

```
  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
```

Tableaux "A" et "B"

```
 1 ||  0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0 ||  10.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0 ||  20.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
```

```
  z =  0.00
  w =  0.00
```

```
fam : i = [1_3];
      ibis = [0_2];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(i) = 10,20,30;
equ : min[som(i <> 1) c2(i >< ibis) * x(i >< ibis)];
      un, #i <> 1, som(j) y(i,j) + x(i ><ibis) = c2(i >< ibis);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1      0      0 !      1      0      0 !      1      0      0

        !  cst          !       code_cst      !       val_cst
  1      2              !      3      0      0 !      2      0      0

        !  variable     !       code_var      !       val_var
  1      1              !      3      0      0 !      2      0      0

mult :
  1     1.000

Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    1      0      0 !      1      0      0 !      1      0      0
Code =    1
```

1er membre :

	no1_ent_som			!	code1_som			!	val1_som		
1	!	3	0	0	!	0	0	0	!	0	0	0
2	!	0	0	0	!	0	0	0	!	0	0	0

	cst1	!	code1_cst			!	val1_cst		
1	! 0	!	0	0	0	!	0	0	0
2	! 0	!	0	0	0	!	0	0	0

	var1	!	code1_var			!	val1_var		
1	! 2	!	0	0	0	!	0	0	0
2	! 1	!	3	0	0	!	2	0	0

	mult1
1	1.00000
2	1.00000

2nd membre :

	no2_ent_som			!	code2_som			!	val2_som		
1	!	0	0	0	!	0	0	0	!	0	0	0

	cst2	!	code2_cst			!	val2_cst		
1	! 2	!	3	0	0	!	2	0	0

	mult2
1	1.00000

Variables :
"x" 1, 0, 0,
"y" 1, 3, 0,
Résultats de la transformation

Tableau "C" :

‖	10.0	20.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	! 0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0

Tableaux "A" et "B"

1	‖ 1.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	‖	10.0
2	‖ 0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	‖	20.0
3	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
      ibis = [0_2];
      j = [1_2];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(j,i) = 10,20,30,40,50,60;
equ : min[som(j,i <> 1) c2(j,i >< ibis) * x(j,i >< ibis)];
      un, #i <> 1, som(j) y(i,j) + som(j) x(j,i><ibis) =
som(j)c2(j,i><ibis);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !     3       1       0 !       0       1       0 !       0       1       0

        !  cst          !       code_cst        !       val_cst
  1      2              !       0       3       0 !       0       2       0

        !  variable     !       code_var        !       val_var
  1      1              !       0       3       0 !       0       2       0

mult :
  1     1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt                val_ptt
    1       0       0 !       1       0       0 !       1       0       0
Code =    1

1er membre :

              no1_ent_som     !     code1_som         !     val1_som
  1   !       3       0       0 !       0       0       0 !       0       0       0
  2   !       3       0       0 !       0       0       0 !       0       0       0

               cst1           !     code1_cst         !     val1_cst
  1   !       0               !       0       0       0 !       0       0       0
  2   !       0               !       0       0       0 !       0       0       0

               var1           !     code1_var         !     val1_var
  1   !       2               !       0       0       0 !       0       0       0
  2   !       1               !       0       3       0 !       0       2       0

     mult1
  1  1.00000
  2  1.00000

 2nd membre :

              no2_ent_som     !     code2_som         !     val2_som
  1   !       3       0       0 !       0       0       0 !       0       0       0

               cst2           !     code2_cst         !     val2_cst
  1   !       2               !       0       3       0 !       0       2       0

     mult2
  1  1.00000
```

```
Variables :
"x"   3,  1,  0,
"y"   1,  3,  0,
   Résultats de la transformation

Tableau "C" :

  ||  10.0 20.0  0.0 40.0 50.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :

  || ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0
Tableaux "A" et "B"

 1 ||  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  1.0  1.0  ||  50.0
 2 ||  0.0  1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ||  70.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00


fam : i = [1_3];
      ibis = [0_2];
      j = [1_2];
      jbis = [0_1];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(j,i) = 10,20,30,40,50,60;
equ : min[som(j<>1,i <> 1) c2(j,i >< ibis) * x(j><jbis,i >< ibis)];
      un, #i<>1, som(j<>1) y(i,j><jbis) + som(j) x(j,i><ibis) =
                 som(j)c2(j,i><ibis);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !      code_som      !      val_som
  1 !    3     1     0 !     1     1     0 !     1     1     0

        !  cst          !      code_cst      !      val_cst
  1      2              !     0     3     0 !     0     2     0

        !  variable     !      code_var      !      val_var
  1      1              !     3     3     0 !     4     2     0

mult :
  1    1.000


Equations :
   equation   1  nom : "un"
      no_ent_ptt          code_ptt          val_ptt
    1     0     0 !     1     0     0 !     1     0     0
Code =   1
```

1er membre :

	no1_ent_som			!	code1_som			!	val1_som		
1	!	3	0	0	!	1	0	0	!	1	0	0
2	!	3	0	0	!	0	0	0	!	0	0	0

	cst1		code1_cst			val1_cst		
1	! 0	!	0	0	0 !	0	0	0
2	! 0	!	0	0	0 !	0	0	0

	var1		code1_var			val1_var		
1	! 2	!	0	3	0 !	0	4	0
2	! 1	!	0	3	0 !	0	2	0

	mult1
1	1.00000
2	1.00000

2nd membre :

	no2_ent_som			!	code2_som			!	val2_som		
1	!	3	0	0	!	0	0	0	!	0	0	0

	cst2		code2_cst			val2_cst		
1	! 2	!	0	3	0 !	0	2	0

	mult2
1	1.00000

Variables :
"x" 3, 1, 0,
"y" 1, 3, 0,
Résultats de la transformation

Tableau "C" :

‖	40.0	50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	! 0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0!	0.0

Tableaux "A" et "B"

1	‖ 1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	‖	50.0
2	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0		70.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

```
fam : i = [1_3];
      ibis = [0_2];
      j = [1_2];
      jbis = [0_1];
cst : c1(i,j) = 1,2,3,4,5,6;
      c2(j,i) = 10,20,30,40,50,60;
equ : min[som(j<>1,i <> 1) c2(j,i >< ibis) * x(j><jbis,i >< ibis)];
      un, #i, som(j<>1) y(i><1,j><jbis) = som(j) c1(i><1,j);
fin :

Fonction objectif :     Il faut minimiser la fonction.
no_elem ! no_ent_som    !       code_som        !       val_som
  1 !    3     1     0 !      1     1     0 !      1     1     0

        !  cst          !       code_cst       !       val_cst
  1      2              !      0     3     0 !      0     2     0

        !  variable     !       code_var       !       val_var
  1      1              !      3     3     0 !      4     2     0

mult :
  1    1.000


Equations :
   equation    1  nom : "un"
      no_ent_ptt          code_ptt              val_ptt
    1     0     0 !      0     0     0 !       0     0     0
Code =    1

1er membre :

            no1_ent_som   !     code1_som       !     val1_som
  1  !      3     0     0 !      1     0     0 !      1     0     0

             cst1         !     code1_cst       !     val1_cst
  1  !      0             !      0     0     0 !      0     0     0

             var1         !     code1_var       !     val1_var
  1  !      2             !      1     3     0 !      1     4     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som   !     code2_som       !     val2_som
  1  !      3     0     0 !      0     0     0 !      0     0     0

             cst2         !     code2_cst       !     val2_cst
  1  !      1             !      1     0     0 !      1     0     0

    mult2
  1  1.00000

Variables :
"x"    3,   1,   0,
"y"    1,   3,   0,
```

Résultats de la transformation

Tableau "C" :

|| 40.0 50.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableau "D" :

|| ! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0! 0.0

Tableaux "A" et "B"

1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	3.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

```
equ : min[3 *  x + 4 * y];
      un, 3 * x + 4 * y;
```

**** erreur (49) du type "=", "<=" ou ">=" attendu ****

```
equ : min[3 *  x + 4 * y];
      un, 3 * x + 4 * y;
                      ^
```

TESTS SUR L'ENCHAINEMENT DES PARAGRAPHES

TRAITEMENT DU FICHIER "A:\CHAIN\CHN000.PAS"

```
fam :

cst : c = 1;

equ : min [c * x];
      un, c * x <= 2;

fin :
```

*** erreur (14) du type "=" attendu ***

```
fam :

cst : c = 1;
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN001.PAS"

```
fam : i = [1_3];

cst :

equ : min [x + y];
      un, x + y <= 2;

fin :
```

*** erreur (42) du type mot reserve ***

```
cst :

equ : min [x + y];
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN002.PAS"

```
fam : i = [1_3];

cst : c = 1;

fin :
```

*** erreur (34) du type paragraphe "equ :" attendu ***

```
cst : c = 1;

fin :
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN003.PAS"

```
fam : i = [1_3];

cst : c = 1;

equ : min [c * x];
      un, 3 * x >= 2;
```

*** erreur (67) du type fin de fichier inattendue ***

```
      un, 3 * x >= 2;
^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN004.PAS"

```
cst : c = 1;

fam : i = [1_3];

equ : min [c * x];
      un, 3 * x >= 2;
```

** erreur (35) du type ordre des paragraphes incoherent **

```
cst : c = 1;

fam : i = [1_3];
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN005.PAS"

```
cst : c = 1;

equ : min [c * x];
      un, 3 * x >= 2;

fam : i = [1_3];

fin :
```

** erreur (35) du type ordre des paragraphes incoherent **

```
      un, 3 * x >= 2;

fam : i = [1_3];
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN006.PAS"

```
equ : min [x];
      un, 3 * x >= 2;

fam : i = [1_3];

cst : c = 1;

fin :
```

** erreur (35) du type ordre des paragraphes incoherent **

```
      un, 3 * x >= 2;

fam : i = [1_3];
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN007.PAS"

```
fin :
```

**** erreur (3) du type nom de paragraphe attendu ****

```
fin :
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN008.PAS"

```
com : fichier inutile !

fin :
```

*** erreur (34) du type paragraphe "equ :" attendu ***

```
com : fichier inutile !
    ^
```

TRAITEMENT DU FICHIER "A:\CHAIN\CHN009.PAS"

```
com : fichier inutile !

equ : min[x + y];
      un, x + y >= 2;

fin :
```

*** erreur (34) du type paragraphe "equ :" attendu ***

```
com : fichier inutile !
    ^
```

TRAITEMENT DU FICHIER "B:\CHAIN\CHN010.PAS"

```
fam : i = [1_3];

cst : c = 12;

fam : j = [1_2];

equ : min[3 * x + 4 * y];
      un, x + y >= 3;

fin :
```

* erreur (35) du type ordre des paragraphes incohérent *

```
cst : c = 12;

fam : j = [1_2];
    ^
```

TRAITEMENT DU FICHIER "B:\CHAIN\CHN011.PAS"

```
cst : c = 12;

cst : k = 13;

equ : min[3 * x + 4 * y];
      un, x + y >= 3;

fin :
```

** erreur (2) du type paragraphe apparaissant deux fois **

```
cst : c = 12;

cst : k = 13;
    ^
```

TRAITEMENT DU FICHIER "B:\CHAIN\CHN012.PAS"

```
fam : i = [1_3];

cst : c = 12;

equ : min[3 * x + 4 * y];
      un, x + y >= 3;

equ : min[3 * x + 4 * y];
      un, x + y >= 3;

fin :
```

** erreur (2) du type paragraphe apparaissant deux fois **

```
      un, x + y >= 3;

equ : min[3 * x + 4 * y];
    ^
```

SIMPLEXE ET ANALYSE DE SENSIBILITE

```
equ : max[ 40 * x1 + 50 * x2];
      un, x1 + x2 <= 50;
      deux, 3 * x1 + 4 * x2 <= 180;
      trois, x2 <= 40;
fin :
```

Résultats de la transformation

Tableau "C" :

‖	-40.0	-50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	50.0
2	‖ 3.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	180.0
3	‖ 0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	40.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

```
  z =  0.00
  w =  0.00
```

Résultats de la transformation

Tableau "C" :

‖	-40.0	-50.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	50.0
2	‖ 3.0	4.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	180.0
3	‖ 0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	40.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

```
  z =  0.00
  w =  0.00

pivotage autour de  3 et  2
```

Résultats de la transformation

Tableau "C" :

‖	-40.0	0.0	0.0	0.0	50.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0		10.0
2	3.0	0.0	0.0	1.0	-4.0	0.0	0.0	0.0	0.0	0.0		20.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0		40.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 2000.00
w = 0.00

pivotage autour de 2 et 1

Résultats de la transformation

Tableau "C" :

	0.0	0.0	0.0	13.3	-3.3	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	-0.3	0.3	0.0	0.0	0.0	0.0	0.0		3.3
2	1.0	0.0	0.0	0.3	-1.3	0.0	0.0	0.0	0.0	0.0		6.7
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0		40.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 2266.67
w = 0.00

pivotage autour de 1 et 5

Résultats de la transformation

Tableau "C" :

	0.0	0.0	10.0	10.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	3.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	10.0
2	1.0	0.0	4.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	20.0
3	0.0	1.0	-3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	30.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 2300.00
w = 0.00

Solution optimale :

Fonction objectif : z = 2300.00000

x1 =	20.00000
x2 =	30.00000
trois,ve+ =	10.00000

Résultats sur les coûts :

x1 :	0.00000
x2 :	0.00000
un,ve+ :	10.00000
deux,ve+ :	10.00000
trois,ve+ :	0.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-10.00000, 2.50000]

avec une pente 20.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-3.33333, 10.00000]

avec une pente 30.00000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-3.33333, 10.00000]

avec une pente -10.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-30.00000, 10.00000]

avec une pente -10.00000

Analyse de sensibilite sur la ligne 3 :

La solution reste valable pour le paramètre DELTA dans
[-10.00000,+∞]

avec une pente 0.00000

```
equ : min[x1 - x2 + x3 + x4 + x5 - x6];
      un, x1 + x4  + 6 * x6 = 9;
      deux, 3 * x1 + x2 - 4 * x3 + 2 * x6 = 2;
      trois, x1 + 2 * x3 + x5 + 2 * x6 = 6;
fin :
```

Résultats de la transformation

Tableau "C" :

1.0	-1.0	1.0	1.0	1.0	-1.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	1.0	0.0	6.0	0.0	0.0	0.0	0.0	9.0
2	3.0	1.0	-4.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	2.0
3	1.0	0.0	2.0	0.0	1.0	2.0	0.0	0.0	0.0	0.0	6.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

1.0	-1.0	1.0	1.0	1.0	-1.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	1.0	0.0	6.0	1.0	0.0	0.0	0.0	9.0
2	3.0	1.0	-4.0	0.0	0.0	2.0	0.0	1.0	0.0	0.0	2.0
3	1.0	0.0	2.0	0.0	1.0	2.0	0.0	0.0	1.0	0.0	6.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
  z =  0.00
  w =  0.00

pivotage autour de  2 et  6

   Résultats de la transformation

Tableau "C" :
  ||   2.5 -0.5 -1.0  1.0  1.0  0.0  0.0  0.5  0.0  0.0

Tableau "D" :
  ||  10.0  4.0 -18.0 -1.0 -1.0  0.0  0.0  5.0  0.0  0.0

Tableaux "A" et "B"
 1 || -8.0 -3.0 12.0  1.0  0.0  0.0  1.0 -3.0  0.0  0.0  ||  3.0
 2 ||  1.5  0.5 -2.0  0.0  0.0  1.0  0.0  0.5  0.0  0.0  ||  1.0
 3 || -2.0 -1.0  6.0  0.0  1.0  0.0  0.0 -1.0  1.0  0.0  ||  4.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

  z =  1.00
  w = -7.00

pivotage autour de  1 et  3

   Résultats de la transformation

Tableau "C" :
  ||   1.8 -0.8  0.0  1.1  1.0  0.0  0.1  0.3  0.0  0.0

Tableau "D" :
  ||  -2.0 -0.5  0.0  0.5 -1.0  0.0  1.5  0.5  0.0  0.0

Tableaux "A" et "B"
 1 || -0.7 -0.3  1.0  0.1  0.0  0.0  0.1 -0.3  0.0  0.0  ||  0.3
 2 ||  0.2  0.0  0.0  0.2  0.0  1.0  0.2  0.0  0.0  0.0  ||  1.5
 3 ||  2.0  0.5  0.0 -0.5  1.0  0.0 -0.5  0.5  1.0  0.0  ||  2.5
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

  z =  1.25
  w = -2.50

pivotage autour de  3 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0 -1.2  0.0  1.5  0.1  0.0  0.5 -0.2 -0.9  0.0
```

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	‖	0.0	-0.1	1.0	-0.1	0.3	0.0	-0.1	-0.1	0.3	0.0	‖ 1.1
2	‖	0.0	-0.0	0.0	0.2	-0.1	1.0	0.2	-0.0	-0.1	0.0	‖ 1.3
3	‖	1.0	0.3	0.0	-0.3	0.5	0.0	-0.3	0.3	0.5	0.0	‖ 1.3
4	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
5	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
6	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
7	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
8	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
9	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
10	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0

z = -1.04
w = 0.00

pivotage autour de 3 et 2

Résultats de la transformation

Tableau "C" :

‖	4.8	0.0	0.0	0.3	2.5	0.0	-0.7	1.0	1.5	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	‖	0.3	0.0	1.0	-0.2	0.5	0.0	-0.2	0.0	0.5	0.0	‖ 1.5
2	‖	0.2	0.0	0.0	0.2	0.0	1.0	0.2	0.0	0.0	0.0	‖ 1.5
3	‖	4.0	1.0	0.0	-1.0	2.0	0.0	-1.0	1.0	2.0	0.0	‖ 5.0
4	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
5	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
6	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
7	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
8	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
9	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0
10	‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖ 0.0

z = 5.00
w = 0.00

Solution optimale :

Fonction objectif : z = 5.00000

x2 = 5.00000
x3 = 1.50000
x6 = 1.50000

Résultats sur les coûts :

x1 :	4.83333
x2 :	0.00000
x3 :	0.00000
x4 :	0.33333
x5 :	2.50000
x6 :	0.00000
un,va :	-0.66667
deux,va :	1.00000
trois,va :	1.50000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-4.83333,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-0.33333, 0.66667]

avec une pente 5.00000

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[-2.00000, 3.00000]

avec une pente 1.50000

Analyse de sensibilite sur la colonne 4 :

La solution reste valable pour le paramètre DELTA dans
[-0.33333,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 5 :

La solution reste valable pour le paramètre DELTA dans
[-2.50000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 6 :

La solution reste valable pour le paramètre DELTA dans
[-4.00000, 2.00000]

avec une pente 1.50000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-9.00000, 5.00000]

avec une pente 0.66667

```
Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -5.00000,+∞]

  avec une pente    -1.00000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.50000,+∞]

  avec une pente    -1.50000


equ : min[2000 * x1 + 3000 *  x2];
      un, 1.6 * x1 + 0.8 * x2 >= 80;
      deux, 0.2 * x1 + 0.4 * x2 >= 20;
      trois, 0.1 * x1 + 0.1 * x2 >= 8;
fin :

   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 || -1.6 -0.8  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -80.0
 2 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -20.0
 3 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   -8.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0

   z =  0.00
   w =  0.00

    Résultats de la transformation

 Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

 Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	-1.6	-0.8	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-80.0
2	-0.2	-0.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
3	-0.1	-0.1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	-8.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

	0.0	2000.0	1250.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.5	-0.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0
2	0.0	-0.3	-0.1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-10.0
3	0.0	-0.0	-0.1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	-3.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -100000.00
w = 0.00

pivotage autour de 2 et 2

Résultats de la transformation

Tableau "C" :

	0.0	0.0	416.7	6666.7	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-0.8	1.7	0.0	0.0	0.0	0.0	0.0	0.0	33.3
2	0.0	1.0	0.4	-3.3	0.0	0.0	0.0	0.0	0.0	0.0	33.3
3	0.0	0.0	-0.0	-0.2	1.0	0.0	0.0	0.0	0.0	0.0	-1.3
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -166666.67
w = 0.00

pivotage autour de 3 et 3

Résultats de la transformation

Tableau "C" :

	0.0	0.0	0.0	5000.0	10000.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	5.0	-20.	0.0	0.0	0.0	0.0	0.0	60.0
2	0.0	1.0	0.0	-5.0	10.0	0.0	0.0	0.0	0.0	0.0	20.0
3	0.0	0.0	1.0	4.0	-24.	0.0	0.0	0.0	0.0	0.0	32.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -180000.00
w = 0.00

Solution optimale :

Fonction objectif : z = -180000.00000

x1 = 60.00000
x2 = 20.00000
un,ve+ = 32.00000

Résultats sur les coûts :

x1 : 0.00000
x2 : 0.00000
un,ve+ : 0.00000
deux,ve+ : 5000.00000
trois,ve+ : 10000.00000

```
Analyse de sensibilite sur la colonne  1 :

La solution reste valable pour le paramètre DELTA dans
  [-500.00000,1000.00000]

 avec une pente    60.00000

Analyse de sensibilite sur la colonne  2 :

La solution reste valable pour le paramètre DELTA dans
  [-1000.00000,1000.00000]

 avec une pente    20.00000

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [ -32.00000,+∞]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -8.00000,    4.00000]

  avec une pente -5000.00000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,    1.33333]

  avec une pente -10000.00000


equ : min[- x1 - x2 + 2 * x3];
      un, x1 + x2 + x3 = 12;
      deux, 2 * x1 + 5 * x2 - 6 * x3 = 10;
      trois, 7 * x1 + 10 * x2 - x3 = 70;
fin :

   Résultats de la transformation

Tableau "C" :
  ||  -1.0 -1.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableaux "A" et "B"
 1 ||  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 2 ||  2.0  5.0 -6.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  10.0
 3 ||  7.0 10.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  70.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

   Résultats de la transformation

Tableau "C" :
   ||  -1.0 -1.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
   ||   0.0  0.0  0.0  1.0  1.0  1.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 2 ||  2.0  5.0 -6.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  ||  10.0
 3 ||  7.0 10.0 -1.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  ||  70.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  2 et  2

   Résultats de la transformation

Tableau "C" :
   ||  -0.6  0.0  0.8  0.0  0.2  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
   ||  -3.6  0.0 -13.2  0.0  3.2  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.6  0.0  2.2  1.0 -0.2  0.0  0.0  0.0  0.0  0.0  ||  10.0
 2 ||  0.4  1.0 -1.2  0.0  0.2  0.0  0.0  0.0  0.0  0.0  ||   2.0
 3 ||  3.0  0.0 11.0  0.0 -2.0  1.0  0.0  0.0  0.0  0.0  ||  50.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
```

```
  z =  2.00
  w = -60.00

pivotage autour de  1 et  3

   Résultats de la transformation

Tableau "C" :
  ||  -0.8  0.0  0.0 -0.4  0.3  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  6.0  2.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.3  0.0  1.0  0.5 -0.1  0.0  0.0  0.0  0.0  0.0  ||   4.5
 2 ||  0.7  1.0  0.0  0.5  0.1  0.0  0.0  0.0  0.0  0.0  ||   7.5
 3 ||  0.0  0.0  0.0 -5.0 -1.0  1.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -1.64
  w =  0.00

pivotage autour de  2 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0  1.1  0.0  0.2  0.4  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  6.0  2.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0 -0.4  1.0  0.3 -0.1  0.0  0.0  0.0  0.0  0.0  ||   1.8
 2 ||  1.0  1.4  0.0  0.8  0.1  0.0  0.0  0.0  0.0  0.0  ||  10.2
 3 ||  0.0  0.0  0.0 -5.0 -1.0  1.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  6.75
  w =  0.00
```

La contrainte n° 3 est redondante.

Solution optimale :

Fonction objectif : z = 6.75000

x1 =	10.25000
x3 =	1.75000
trois,va =	0.00000

Résultats sur les coûts :

x1 :	0.00000
x2 :	1.12500
x3 :	0.00000
un,va :	0.25000
deux,va :	0.37500
trois,va :	0.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 0.33333]

avec une pente 10.25000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-1.12500,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[-3.00000, 1.00000]

avec une pente 1.75000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-7.00000, 0.00000]

avec une pente -0.25000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-82.00000, 0.00000]

avec une pente -0.37500

```
equ : min[x1 + 0.1 * x2 - 2 * x3 + 2 * x4];
      un, 10 * x1 - x2 + 3 * x3 + x4 = 0;
      deux, 20 * x1 + 2 * x2 - 21 * x3 - x4 = 0;
      trois, 10 * x1 + x2 = 2;
fin :
```

Résultats de la transformation

Tableau "C" :

1.0	0.1	-2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	10.0	-1.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
2	20.0	2.0	-21.	-1.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
3	10.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		2.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

1.0	0.1	-2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	10.0	-1.0	3.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0		0.0
2	20.0	2.0	-21.	-1.0	0.0	1.0	0.0	0.0	0.0	0.0		0.0
3	10.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0		2.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

0.0	0.2	-2.3	1.9	-0.1	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	-6.0	30.0	4.0	4.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	-0.1	0.3	0.1	0.1	0.0	0.0	0.0	0.0	0.0	‖	0.0
2	0.0	4.0	-27.	-3.0	-2.0	1.0	0.0	0.0	0.0	0.0		0.0
3	0.0	2.0	-3.0	-1.0	-1.0	0.0	1.0	0.0	0.0	0.0		2.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = -2.00

pivotage autour de 2 et 2

Résultats de la transformation

Tableau "C" :

‖	0.0	0.0	-1.0	2.0	0.0	-0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	-10.5	-0.5	1.0	1.5	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
2	0.0	1.0	-6.7	-0.7	-0.5	0.3	0.0	0.0	0.0	0.0		0.0
3	0.0	0.0	10.5	0.5	0.0	-0.5	1.0	0.0	0.0	0.0		2.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = -2.00

pivotage autour de 3 et 3

Résultats de la transformation

Tableau "C" :

‖	0.0	0.0	0.0	2.1	0.0	-0.1	0.1	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.1
2	0.0	1.0	0.0	-0.4	-0.5	-0.1	0.6	0.0	0.0	0.0		1.3
3	0.0	0.0	1.0	0.0	0.0	-0.0	0.1	0.0	0.0	0.0		0.2
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.18
w = 0.00

Solution optimale :

Fonction objectif : z = 0.18095

x1 = 0.07143
x2 = 1.28571
x3 = 0.19048

Résultats sur les coûts :

x1 : 0.00000
x2 : 0.00000
x3 : 0.00000
x4 : 2.09524
un,va : 0.00000
deux,va : -0.09524
trois,va : 0.09048

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-13.33333, 2.53333]

avec une pente 0.07143

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-4.88889, 0.14074]

avec une pente 1.28571

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 0.95000]

avec une pente 0.19048

Analyse de sensibilite sur la colonne 4 :

La solution reste valable pour le paramètre DELTA dans
[-2.09524,+∞]

avec une pente 0.00000

```
Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.42857,    2.57143]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [ -10.00000,    4.00000]

  avec une pente     0.09524

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,+∞]

  avec une pente    -0.09048


equ : max[4 * x + 5 * y];
      un, 2 * x + 3 * y <= 12;
      deux, y <= 3;
      trois, x <= 5;
fin :


   Résultats de la transformation

Tableau "C" :
  ||  -4.0 -5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  2.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  12.0
 2 ||  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   3.0
 3 ||  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   5.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

  z =  0.00
  w =  0.00

   Résultats de la transformation

Tableau "C" :
  ||  -4.0 -5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	2.0	3.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	12.0
2	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	5.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 2 et 2

Résultats de la transformation

Tableau "C" :

-4.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	2.0	0.0	1.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0
2	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	5.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 15.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

0.0	0.0	2.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.5	-1.5	0.0	0.0	0.0	0.0	0.0	0.0	1.5
2	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0
3	0.0	0.0	-0.5	1.5	1.0	0.0	0.0	0.0	0.0	0.0	3.5
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 21.00
w = 0.00

pivotage autour de 3 et 4

Résultats de la transformation

Tableau "C" :

0.0	0.0	1.7	0.0	0.7	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	5.0
2	0.0	1.0	0.3	0.0	-0.7	0.0	0.0	0.0	0.0	0.0	0.7
3	0.0	0.0	-0.3	1.0	0.7	0.0	0.0	0.0	0.0	0.0	2.3
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 23.33
w = 0.00

Solution optimale :

Fonction objectif : z = 23.33333

x =	5.00000
y =	0.66667
deux,ve+ =	2.33333

Résultats sur les coûts :

x :	0.00000
y :	0.00000
un,ve+ :	1.66667
deux,ve+ :	0.00000
trois,ve+ :	0.66667

```
Analyse de sensibilite sur la colonne  1 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,    0.66667]

 avec une pente     5.00000

Analyse de sensibilite sur la colonne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.00000,    5.00000]

 avec une pente     0.66667

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,    7.00000]

  avec une pente    -1.66667

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.33333,+∞]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -3.50000,    1.00000]

  avec une pente    -0.66667


equ : max[- x + y];
      un, x >= 1;
      deux, - x + y <= 2;
fin :


   Résultats de la transformation

Tableau "C" :
  ||   1.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	1.0
2	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	2.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

‖	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	1.0
2	-1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	2.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

‖	0.0	-1.0	1.0	0.0	-1.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	1.0
2	0.0	1.0	-1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	3.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

```
  z = -1.00
  w =  0.00

pivotage autour de  2 et  2

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  0.0 -1.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0 ||  1.0
 2 ||  0.0  1.0 -1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0 ||  3.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0

  z =  2.00
  w =  0.00


   Solution optimale :

         Fonction objectif : z =     2.00000

                                        x =     1.00000
                                        y =     3.00000

   Résultats sur les coûts :

                                        x :     0.00000
                                        y :     0.00000
                                  un,ve+ :     0.00000
                                 deux,ve+ :    1.00000
                                    un,va :    0.00000

   Analyse de sensibilite sur la colonne  1 :

   La solution reste valable pour le paramètre DELTA dans
     [-∞,+∞]

    avec une pente     1.00000

   Analyse de sensibilite sur la colonne  2 :

   La solution reste valable pour le paramètre DELTA dans
     [-∞,    1.00000]

    avec une pente     3.00000
```

```
Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.00000,+∞]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -3.00000,+∞]

  avec une pente    -1.00000


equ : max[y];
      un, x + 2 * y >= 3;
      deux, 3 * x + y >= 3;
      trois, x + y <= 3;
fin :

   Résultats de la transformation

Tableau "C" :
  ||  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  2.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  3.0
 2 ||  1.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  3.0
 3 ||  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  3.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

   z =  0.00
   w =  0.00

   Résultats de la transformation

Tableau "C" :
  ||  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	2.0	1.0	-1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0		3.0
2	1.0	3.0	0.0	-1.0	0.0	0.0	1.0	0.0	0.0	0.0		3.0
3	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0		3.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

pivotage autour de 2 et 2

Résultats de la transformation

Tableau "C" :

-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

-1.7	0.0	1.0	-0.3	0.0	0.0	1.3	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.7	0.0	-1.0	0.3	0.0	1.0	-0.3	0.0	0.0	0.0		2.0
2	0.3	1.0	0.0	-0.3	0.0	0.0	0.3	0.0	0.0	0.0		1.0
3	0.7	0.0	0.0	0.3	1.0	0.0	-0.3	0.0	0.0	0.0		2.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = -2.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

0.0	0.0	-0.6	0.2	0.0	0.6	-0.2	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-0.6	0.2	0.0	0.6	-0.2	0.0	0.0	0.0	1.2
2	0.0	1.0	0.2	-0.4	0.0	-0.2	0.4	0.0	0.0	0.0	0.6
3	0.0	0.0	0.4	0.2	1.0	-0.4	-0.2	0.0	0.0	0.0	1.2
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 1.20
w = 0.00

pivotage autour de 3 et 3

Résultats de la transformation

Tableau "C" :

0.0	0.0	0.0	0.5	1.5	0.0	-0.5	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.5	1.5	0.0	-0.5	0.0	0.0	0.0	3.0
2	0.0	1.0	0.0	-0.5	-0.5	0.0	0.5	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.5	2.5	-1.0	-0.5	0.0	0.0	0.0	3.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 3.00
w = 0.00

Solution optimale :

Fonction objectif : z = 3.00000

y =	3.00000
x =	0.00000
un,ve+ =	3.00000

Résultats sur les coûts :

y :	0.00000
x :	0.00000
un,ve+ :	0.00000
un,ve- :	0.50000
trois,ve+ :	1.50000
un,va :	0.00000
deux,va :	-0.50000

```
Analyse de sensibilite sur la colonne  1 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,    1.00000]

 avec une pente     3.00000

Analyse de sensibilite sur la colonne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.00000,+∞]

 avec une pente     0.00000

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,    3.00000]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [   0.00000,    6.00000]

  avec une pente     0.50000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.20000,    0.00000]

  avec une pente    -1.50000


equ : max[y];
      un, - x + y >= 0.2;
      deux, x + y >= 2;
      trois, y >= 1.1;
      quatre, y <= 2;
fin :


   Résultats de la transformation

Tableau "C" :
  ||  -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.2
2	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		2.0
3	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		1.1
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		2.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	1.0	-1.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0		0.2
2	1.0	1.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0		2.0
3	1.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0		1.1
4	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0		2.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

0.0	-1.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

Tableau "D" :

0.0	-3.0	-2.0	1.0	1.0	0.0	3.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	-1.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0		0.2
2	0.0	2.0	1.0	-1.0	0.0	0.0	-1.0	1.0	0.0	0.0		1.8
3	0.0	1.0	1.0	0.0	-1.0	0.0	-1.0	0.0	1.0	0.0		0.9
4	0.0	1.0	1.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0		1.8
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0		0.0

```
z =  0.20
w = -2.70

pivotage autour de  3 et  2

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0  0.0  0.0 -1.0  0.0  0.0  0.0  1.0  0.0

Tableau "D" :
  ||   0.0  0.0  1.0  1.0 -2.0  0.0  0.0  0.0  3.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  0.0  0.0  0.0 -1.0  0.0  0.0  0.0  1.0  0.0 ||  1.1
 2 ||  0.0  0.0 -1.0 -1.0  2.0  0.0  1.0  1.0 -2.0  0.0 ||  0.0
 3 ||  0.0  1.0  1.0  0.0 -1.0  0.0 -1.0  0.0  1.0  0.0 ||  0.9
 4 ||  0.0  0.0  0.0  0.0  1.0  1.0  0.0  0.0 -1.0  0.0 ||  0.9
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0

z =  1.10
w =  0.00

pivotage autour de  2 et  5

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0 -0.5 -0.5  0.0  0.0  0.5  0.5  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  1.0  1.0  1.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  0.0 -0.5 -0.5  0.0  0.0  0.5  0.5  0.0  0.0 ||  1.1
 2 ||  0.0  0.0 -0.5 -0.5  1.0  0.0  0.5  0.5 -1.0  0.0 ||  0.0
 3 ||  0.0  1.0  0.5 -0.5  0.0  0.0 -0.5  0.5  0.0  0.0 ||  0.9
 4 ||  0.0  0.0  0.5  0.5  0.0  1.0 -0.5 -0.5  0.0  0.0 ||  0.9
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0

z =  1.10
w =  0.00

pivotage autour de  3 et  3

   Résultats de la transformation

Tableau "C" :
  ||   0.0  1.0  0.0 -1.0  0.0  0.0  0.0  1.0  0.0  0.0
```

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	1.0	1.0	0.0	-1.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0
2	0.0	1.0	0.0	-1.0	1.0	0.0	0.0	1.0	-1.0	0.0	0.9
3	0.0	2.0	1.0	-1.0	0.0	0.0	-1.0	1.0	0.0	0.0	1.8
4	0.0	-1.0	0.0	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 2.00
w = 0.00

pivotage autour de 4 et 4

Résultats de la transformation

Tableau "C" :

0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	2.0
2	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	-1.0	0.0	0.9
3	0.0	1.0	1.0	0.0	0.0	1.0	-1.0	0.0	0.0	0.0	1.8
4	0.0	-1.0	0.0	1.0	0.0	1.0	0.0	-1.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 2.00
w = 0.00

Solution optimale :

Fonction objectif : z = 2.00000

y =	2.00000
un,ve+ =	1.80000
un,ve- =	0.00000
deux,ve- =	0.90000

Résultats sur les coûts :

y :	0.00000
x :	0.00000
un,ve+ :	0.00000
un,ve- :	0.00000
deux,ve- :	0.00000
quatre,ve+ :	1.00000
un,va :	0.00000
deux,va :	0.00000
trois,va :	0.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 1.00000]

avec une pente 2.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 1.80000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 0.00000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 3 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 0.90000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 4 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente -1.00000

```
equ : min[2 * x + 3 * y];
      un, x + 2 * y >= 12;
      deux, 3 * x + 4 * y >= 30;
      trois, 3 * x + y >= 15;
fin :
```

Résultats de la transformation

Tableau "C" :

‖	2.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ -1.0	-2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-12.0
2	‖ -3.0	-4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-30.0
3	‖ -3.0	-1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-15.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

‖	2.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ -1.0	-2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-12.0
2	‖ -3.0	-4.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-30.0
3	‖ -3.0	-1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	-15.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

pivotage autour de 2 et 1

Résultats de la transformation

Tableau "C" :

‖	0.0	0.3	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0

```
Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0 -0.7  1.0 -0.3  0.0  0.0  0.0  0.0  0.0  0.0  ||  -2.0
 2 ||  1.0  1.3  0.0 -0.3  0.0  0.0  0.0  0.0  0.0  0.0  ||  10.0
 3 ||  0.0  3.0  0.0 -1.0  1.0  0.0  0.0  0.0  0.0  0.0  ||  15.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -20.00
  w =  0.00

pivotage autour de  1 et  2

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0  0.5  0.5  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0  1.0 -1.5  0.5  0.0  0.0  0.0  0.0  0.0  0.0  ||   3.0
 2 ||  1.0  0.0  2.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   6.0
 3 ||  0.0  0.0  4.5 -2.5  1.0  0.0  0.0  0.0  0.0  0.0  ||   6.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -21.00
  w =  0.00


   Solution optimale :

         Fonction objectif : z =  -21.00000

                                        x =      6.00000
                                        y =      3.00000
                             trois,ve+ =      6.00000

  Résultats sur les coûts :

                                        x :      0.00000
                                        y :      0.00000
                                  un,ve+ :      0.50000
                                 deux,ve+ :      0.50000
                                trois,ve+ :      0.00000
```

```
Analyse de sensibilite sur la colonne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -0.50000,    0.25000]

 avec une pente     6.00000

Analyse de sensibilite sur la colonne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -0.33333,    1.00000]

 avec une pente     3.00000

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.33333,    2.00000]

  avec une pente    -0.50000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -6.00000,    2.40000]

  avec une pente    -0.50000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [  -6.00000,+∞]

  avec une pente     0.00000


equ : min[3 * x + 2 * y];
      un, x + y <= 8;
      deux, 6 * x + 4 * y >= 12;
      trois, 5 * x + 8 * y = 20;
fin :

   Résultats de la transformation

Tableau "C" :
  ||   3.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0
2	6.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	12.0
3	5.0	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.0
2	6.0	4.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	12.0
3	5.0	8.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	20.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 3 et 2

Résultats de la transformation

Tableau "C" :

1.8	0.0	0.0	0.0	0.0	-0.3	0.0	0.0	0.0	0.0

Tableau "D" :

-3.5	0.0	0.0	1.0	0.0	1.5	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.4	0.0	1.0	0.0	0.0	-0.1	0.0	0.0	0.0	0.0	5.5
2	3.5	0.0	0.0	-1.0	1.0	-0.5	0.0	0.0	0.0	0.0	2.0
3	0.6	1.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	2.5
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
  z = -5.00
  w = -2.00

pivotage autour de  2 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0   0.0   0.0   0.5  -0.5   0.0   0.0   0.0   0.0   0.0

Tableau "D" :
  ||   0.0   0.0   0.0   0.0   1.0   1.0   0.0   0.0   0.0   0.0

Tableaux "A" et "B"
 1 ||  0.0   0.0   1.0   0.1  -0.1  -0.1   0.0   0.0   0.0   0.0  ||  5.3
 2 ||  1.0   0.0   0.0  -0.3   0.3  -0.1   0.0   0.0   0.0   0.0  ||  0.6
 3 ||  0.0   1.0   0.0   0.2  -0.2   0.2   0.0   0.0   0.0   0.0  ||  2.1
 4 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
 5 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
 6 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
 7 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
 8 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
 9 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0
10 ||  0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  ||  0.0

  z = -6.00
  w =  0.00


  Solution optimale :

        Fonction objectif : z =    -6.00000

                                          x =     0.57143
                                          y =     2.14286
                                  un,ve+ =     5.28571

  Résultats sur les coûts :

                                          x :     0.00000
                                          y :     0.00000
                                  un,ve+ :     0.00000
                                 deux,ve+ :     0.50000
                                 deux,ve- :    -0.50000
                                  deux,va :     0.00000

  Analyse de sensibilite sur la colonne  1 :

  La solution reste valable pour le paramètre DELTA dans
    [  -1.75000,+∞]

   avec une pente     0.57143

  Analyse de sensibilite sur la colonne  2 :

  La solution reste valable pour le paramètre DELTA dans
    [-∞,    2.80000]

   avec une pente     2.14286
```

```
Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -5.28571,+∞]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,   12.00000]

  avec une pente     0.50000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [ -10.00000,    4.00000]

  avec une pente     0.00000


equ : max[2 * x + 5 * y];
      un, x + y >= 4;
      deux, x + 2 * y <= 6;
fin :


   Résultats de la transformation

Tableau "C" :
  ||  -2.0 -5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  4.0
 2 ||  1.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  6.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  0.0

   z =  0.00
   w =  0.00

    Résultats de la transformation

 Tableau "C" :
   ||  -2.0 -5.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

 Tableau "D" :
   ||   0.0  0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	1.0	1.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	4.0
2	1.0	2.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

0.0	-3.0	-2.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	1.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	4.0
2	0.0	1.0	1.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	2.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 8.00
w = 0.00

pivotage autour de 2 et 2

Résultats de la transformation

Tableau "C" :

0.0	0.0	1.0	3.0	-1.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	-2.0	-1.0	2.0	0.0	0.0	0.0	0.0	0.0	2.0
2	0.0	1.0	1.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	2.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 14.00
w = 0.00

Solution optimale :

Fonction objectif : z = 14.00000

x = 2.00000
y = 2.00000

Résultats sur les coûts :

x : 0.00000
y : 0.00000
un,ve+ : 1.00000
deux,ve+ : 3.00000
un,va : -1.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-0.50000,+∞]

avec une pente 2.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 1.00000]

avec une pente 2.00000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-1.00000, 2.00000]

avec une pente 1.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-2.00000, 2.00000]

avec une pente -3.00000

```
equ : max[9 * x + 4 * y];
      un, x + y >= 4;
      deux, 2 * x + y <= 6;
fin :
```

Résultats de la transformation

Tableau "C" :

-9.0	-4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
2	2.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

-9.0	-4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	1.0	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	4.0
2	2.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	6.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 2 et 1

Résultats de la transformation

Tableau "C" :

0.0	0.5	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	-0.5	1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.5	-1.0	-0.5	1.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.5	0.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	3.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 27.00
w = -1.00

pivotage autour de 1 et 2

Résultats de la transformation

Tableau "C" :

0.0	0.0	1.0	5.0	-1.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	1.0	-2.0	-1.0	2.0	0.0	0.0	0.0	0.0	0.0	2.0
2	1.0	0.0	1.0	1.0	-1.0	0.0	0.0	0.0	0.0	0.0	2.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 26.00
w = 0.00

Solution optimale :

Fonction objectif : z = 26.00000

x = 2.00000
y = 2.00000

Résultats sur les coûts :

x : 0.00000
y : 0.00000
un,ve+ : 1.00000
deux,ve+ : 5.00000
un,va : -1.00000

```
Analyse de sensibilite sur la colonne  1 :

La solution reste valable pour le paramètre DÉLTA dans
  [-∞,    1.00000]

 avec une pente     2.00000

Analyse de sensibilite sur la colonne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -0.50000,+∞]

 avec une pente     2.00000

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [  -1.00000,    2.00000]

   avec une pente     1.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,    2.00000]

   avec une pente    -5.00000


equ : min [ 3 * x + 2 * y ];
      un, x + y <= 2;
      deux, x+ y >= 3;
fin :


   Résultats de la transformation

Tableau "C" :
  ||   3.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  2.0
 2 ||  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  3.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  0.0

   z =  0.00
   w =  0.00
```

Résultats de la transformation

Tableau "C" :

‖	3.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	2.0
2	‖ 1.0	1.0	0.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	3.0
3	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 1

Résultats de la transformation

Tableau "C" :

‖	0.0	-1.0	-3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	2.0
2	‖ 0.0	0.0	-1.0	-1.0	1.0	0.0	0.0	0.0	0.0	0.0	‖	1.0
3	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
4	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = -6.00
w = -1.00

**** erreur (75) du type execution les contraintes sont incompatibles*

**

```
fam : i = [1_2];
cst : c1(i) = 2000, 3000;
      c2(i) = 1.6, 0.8;
      c3(i) = 0.2, 0.4;
      #i, c4(i) = 0.1;
equ : min [som(i) c1(i) * x(i)];
      un, som(i) c2(i) * x(i) >= 80;
      deux, som(i) c3(i) * x(i) >= 20;
      trois, som(i) c4(i) * x(i) >= 8;
com : meme probleme que sim002
fin :
```

Résultats de la transformation

Tableau "C" :

2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	-1.6	-0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-80.0
2	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
3	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :

2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	-1.6	-0.8	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-80.0
2	-0.2	-0.4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
3	-0.1	-0.1	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	-8.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
 z =  0.00
 w =  0.00

pivotage autour de  1 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0 2000.0 1250.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  0.5 -0.6  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  50.0
 2 ||  0.0 -0.3 -0.1  1.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -10.0
 3 ||  0.0 -0.0 -0.1  0.0  1.0  0.0  0.0  0.0  0.0  0.0 ||  -3.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

 z = -100000.00
 w =  0.00

pivotage autour de  2 et  2

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0 416.7 6666.7  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  0.0 -0.8  1.7  0.0  0.0  0.0  0.0  0.0  0.0 ||  33.3
 2 ||  0.0  1.0  0.4 -3.3  0.0  0.0  0.0  0.0  0.0  0.0 ||  33.3
 3 ||  0.0  0.0 -0.0 -0.2  1.0  0.0  0.0  0.0  0.0  0.0 ||  -1.3
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

 z = -166666.67
 w =  0.00

pivotage autour de  3 et  3

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0  0.0 5000.0 10000.0  0.0  0.0  0.0  0.0  0.0
```

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	1.0	0.0	0.0	5.0	-20.	0.0	0.0	0.0	0.0	0.0	60.0
2	0.0	1.0	0.0	-5.0	10.0	0.0	0.0	0.0	0.0	0.0	20.0
3	0.0	0.0	1.0	4.0	-24.	0.0	0.0	0.0	0.0	0.0	32.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -180000.00
w = 0.00

Solution optimale :

Fonction objectif : z = -180000.00000

x(1) = 60.00000
x(2) = 20.00000
un,ve+ = 32.00000

Résultats sur les coûts :

x(1) : 0.00000
x(2) : 0.00000
un,ve+ : 0.00000
deux,ve+ : 5000.00000
trois,ve+ : 10000.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-500.00000,1000.00000]

avec une pente 60.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-1000.00000,1000.00000]

avec une pente 20.00000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-32.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-8.00000, 4.00000]

avec une pente -5000.00000

```
Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
   [  -2.00000,    1.33333]

   avec une pente -10000.00000


fam : i = [1_2];
      j = [1_3];
cst : c1(i) = 2000, 3000;
      c2(i,j) = 1,2,3,4,5,6;
      c3(i) = 0.2, 0.4;
      #i, c4(i) = 0.1;
equ : min [som(i) c1(i) * x(i)];
      un, #j, som(i) c2(i,j) * y(i,j) >= 80;
      deux, som(i) c3(i) * x(i) >= 20;
      trois, som(i) c4(i) * x(i) >= 8;
fin :

   Résultats de la transformation

Tableau "C" :
   ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
   ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0  0.0 -1.0  0.0  0.0 -4.0  0.0  0.0  0.0  0.0  ||  -80.0
 2 ||  0.0  0.0  0.0 -2.0  0.0  0.0 -5.0  0.0  0.0  0.0  ||  -80.0
 3 ||  0.0  0.0  0.0  0.0 -3.0  0.0  0.0 -6.0  0.0  0.0  ||  -80.0
 4 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -20.0
 5 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   -8.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0

   z =  0.00
   w =  0.00

    Résultats de la transformation

Tableau "C" :
   ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
   ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableaux "A" et "B"
 1 ||  0.0  0.0 -1.0  0.0  0.0 -4.0  0.0  0.0  1.0  0.0 ||  -80.0
 2 ||  0.0  0.0  0.0 -2.0  0.0  0.0 -5.0  0.0  0.0  1.0 ||  -80.0
 3 ||  0.0  0.0  0.0  0.0 -3.0  0.0  0.0 -6.0  0.0  0.0 ||  -80.0
 4 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -20.0
 5 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   -8.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0

  z =  0.00
  w =  0.00

pivotage autour de  1 et  3

   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0  0.0  1.0  0.0  0.0  4.0  0.0  0.0 -1.0  0.0 ||   80.0
 2 ||  0.0  0.0  0.0 -2.0  0.0  0.0 -5.0  0.0  0.0  1.0 ||  -80.0
 3 ||  0.0  0.0  0.0  0.0 -3.0  0.0  0.0 -6.0  0.0  0.0 ||  -80.0
 4 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -20.0
 5 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   -8.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||    0.0

  z =  0.00
  w =  0.00

pivotage autour de  2 et  4

   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	-6.0	0.0	0.0	-80.0
4	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
5	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 3 et 5

Résultats de la transformation

Tableau "C" :

2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	26.7
4	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
5	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = 0.00
w = 0.00

pivotage autour de 4 et 2

Résultats de la transformation

Tableau "C" :

500.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	26.7
4	0.5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0
5	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -150000.00
w = 0.00

pivotage autour de 5 et 1

Résultats de la transformation

Tableau "C" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	26.7
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.0
5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -180000.00
w = 0.00

Solution optimale :

Fonction objectif : z = -180000.00000

x(1) = 60.00000
x(2) = 20.00000
y(1,1) = 80.00000
y(1,2) = 40.00000
y(1,3) = 26.66667

Résultats sur les coûts :

x(1)	:	0.00000
x(2)	:	0.00000
y(1,1)	:	0.00000
y(1,2)	:	0.00000
y(1,3)	:	0.00000
y(2,1)	:	0.00000
y(2,2)	:	0.00000
y(2,3)	:	0.00000
un(1),ve+	:	0.00000
un(2),ve+	:	0.00000
un(3),ve+	:	0.00000
deux,ve+	:	5000.00000
trois,ve+	:	10000.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-500.00000,1000.00000]

avec une pente 60.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-1000.00000,1000.00000]

avec une pente 20.00000

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 80.00000

Analyse de sensibilite sur la colonne 4 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 40.00000

Analyse de sensibilite sur la colonne 5 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 26.66667

Analyse de sensibilite sur la colonne 6 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

```
Analyse de sensibilite sur la colonne  7 :

La solution reste valable pour le paramètre DELTA dans
  [    0.00000,+∞]

 avec une pente     0.00000

Analyse de sensibilite sur la colonne  8 :

La solution reste valable pour le paramètre DELTA dans
  [    0.00000,+∞]

 avec une pente     0.00000

Analyse de sensibilite sur la ligne  1 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,   80.00000]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  2 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,   80.00000]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,   80.00000]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  4 :

La solution reste valable pour le paramètre DELTA dans
  [ -12.00000,    4.00000]

  avec une pente -5000.00000

Analyse de sensibilite sur la ligne  5 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,    3.00000]

  avec une pente -10000.00000


equ : max [x + y];
      un, x + y >= 12;
fin :

   Résultats de la transformation

Tableau "C" :
  ||  -1.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

   Résultats de la transformation

Tableau "C" :
  ||  -1.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0  1.0 -1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  1 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0 -1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableaux "A" et "B"
 1 ||  1.0  1.0 -1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  12.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = 12.00
  w =  0.00

*** erreur (77) du type execution solutions optimales non bornées ***


equ : min[3 * x + 4 * y];
      un, - x + y >= 1;
      deux, x - 2 * y >= 2;
fin :

   Résultats de la transformation

Tableau "C" :
  ||   3.0  4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  1.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -1.0
 2 || -1.0  2.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -2.0
 3 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

   Résultats de la transformation

Tableau "C" :
  ||   3.0  4.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
Tableaux "A" et "B"
 1 ||   1.0 -1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -1.0
 2 ||  -1.0  2.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -2.0
 3 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  2 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0 10.0  0.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableaux "A" et "B"
 1 ||   0.0  1.0  1.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -3.0
 2 ||   1.0 -2.0  0.0 -1.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   2.0
 3 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 4 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 5 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 6 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 7 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 8 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
 9 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0
10 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   0.0

  z = -6.00
  w =  0.00

*** erreur (76) du type execution pas de solution réalisable ***
```

```
fam : i = [1_2];
      j = (bruxelles,liege,namur);
cst : c1(i) = 2000, 3000;
      c2(i,j) = 1,2,3,4,5,6;
      c3(i) = 0.2, 0.4;
      #i, c4(i) = 0.1;
equ : min [som(i) c1(i) * x(i)];
      un, #j, som(i) c2(i,j) * y(i,j) >= 80;
      deux, som(i) c3(i) * x(i) >= 20;
      trois, som(i) c4(i) * x(i) >= 8;
fin :
```

Résultats de la transformation

Tableau "C" :
|| 2000.0 3000.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableau "D" :
|| 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableaux "A" et "B"

1	0.0	0.0	-1.0	0.0	0.0	-4.0	0.0	0.0	0.0	0.0	-80.0
2	0.0	0.0	0.0	-2.0	0.0	0.0	-5.0	0.0	0.0	0.0	-80.0
3	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	-6.0	0.0	0.0	-80.0
4	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
5	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

z = 0.00
w = 0.00

Résultats de la transformation

Tableau "C" :
|| 2000.0 3000.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableau "D" :
|| 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableaux "A" et "B"

1	0.0	0.0	-1.0	0.0	0.0	-4.0	0.0	0.0	1.0	0.0	-80.0
2	0.0	0.0	0.0	-2.0	0.0	0.0	-5.0	0.0	0.0	1.0	-80.0
3	0.0	0.0	0.0	0.0	-3.0	0.0	0.0	-6.0	0.0	0.0	-80.0
4	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
5	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

z = 0.00
w = 0.00

pivotage autour de 1 et 3

Résultats de la transformation

Tableau "C" :
|| 2000.0 3000.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

Tableau "D" :
|| 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

```
Tableaux "A" et "B"
 1 ||  0.0  0.0  1.0  0.0  0.0  4.0  0.0  0.0 -1.0  0.0 ||  80.0
 2 ||  0.0  0.0  0.0 -2.0  0.0  0.0 -5.0  0.0  0.0  1.0 || -80.0
 3 ||  0.0  0.0  0.0  0.0 -3.0  0.0  0.0 -6.0  0.0  0.0 || -80.0
 4 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 || -20.0
 5 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -8.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  2 et  4

   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0  0.0  1.0  0.0  0.0  4.0  0.0  0.0 -1.0  0.0 ||  80.0
 2 ||  0.0  0.0  0.0  1.0  0.0  0.0  2.5  0.0  0.0 -0.5 ||  40.0
 3 ||  0.0  0.0  0.0  0.0 -3.0  0.0  0.0 -6.0  0.0  0.0 || -80.0
 4 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 || -20.0
 5 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -8.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  3 et  5

   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	26.7
4	-0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-20.0
5	-0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-8.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

z = 0.00
w = 0.00

pivotage autour de 4 et 2

Résultats de la transformation

Tableau "C" :

	500.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	26.7
4	0.5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	50.0
5	-0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-3.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

z = -150000.00
w = 0.00

pivotage autour de 5 et 1

Résultats de la transformation

Tableau "C" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	1.0	0.0	0.0	4.0	0.0	0.0	-1.0	0.0	‖	80.0
2	0.0	0.0	0.0	1.0	0.0	0.0	2.5	0.0	0.0	-0.5	‖	40.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	0.0	‖	26.7
4	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	20.0
5	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	60.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = -180000.00
w = 0.00

Solution optimale :

Fonction objectif : z = -180000.00000

x(1)	=	60.00000
x(2)	=	20.00000
y(1,bruxelles)	=	80.00000
y(1,liege)	=	40.00000
y(1,namur)	=	26.66667

Résultats sur les coûts :

x(1)	:	0.00000
x(2)	:	0.00000
y(1,bruxelles)	:	0.00000
y(1,liege)	:	0.00000
y(1,namur)	:	0.00000
y(2,bruxelles)	:	0.00000
y(2,liege)	:	0.00000
y(2,namur)	:	0.00000
un(bruxelles),ve+	:	0.00000
un(liege),ve+	:	0.00000
un(namur),ve+	:	0.00000
deux,ve+	:	5000.00000
trois,ve+	:	10000.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-500.00000,1000.00000]

avec une pente 60.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-1000.00000,1000.00000]

avec une pente 20.00000

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 80.00000

Analyse de sensibilite sur la colonne 4 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 40.00000

Analyse de sensibilite sur la colonne 5 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 26.66667

Analyse de sensibilite sur la colonne 6 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 7 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 8 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 80.00000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 80.00000]

avec une pente 0.00000

```
Analyse de sensibilite sur la ligne  3 :

La solution reste valable pour le paramètre DELTA dans
  [-∞,  80.00000]

  avec une pente     0.00000

Analyse de sensibilite sur la ligne  4 :

La solution reste valable pour le paramètre DELTA dans
  [ -12.00000,    4.00000]

  avec une pente -5000.00000

Analyse de sensibilite sur la ligne  5 :

La solution reste valable pour le paramètre DELTA dans
  [  -2.00000,    3.00000]

  avec une pente -10000.00000
```

```
fam : i = [1_2];
      j = (bruxelles,liege,namur);
cst : c1(i) = 2000, 3000;
      c2(i,j) = 1,2,3,4,5,6;
      c3(i) = 0.2, 0.4;
      #i, c4(i) = 0.1;
equ : min [som(i) c1(i) * x(i)];
      un, #j<>liege, som(i<>1) c2(i,j) * y(i,j) >= 80;
      deux, som(i) c3(i) * x(i) >= 20;
      trois, som(i) c4(i) * x(i) >= 8;
fin :
```

```
   Résultats de la transformation

Tableau "C" :
  ||  2000.0 3000.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||    0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
Tableaux "A" et "B"
 1 ||   0.0  0.0  0.0  0.0  0.0 -4.0  0.0  0.0  0.0  0.0  ||  -80.0
 2 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0 -6.0  0.0  0.0  ||  -80.0
 3 ||  -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||  -20.0
 4 ||  -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||   -8.0
 5 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 6 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 7 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 8 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
 9 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0
10 ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  ||    0.0

  z =   0.00
  w =   0.00
```

Résultats de la transformation

Tableau "C" :

‖	2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 0.0	0.0	0.0	0.0	0.0	-4.0	0.0	0.0	1.0	0.0	‖	-80.0
2	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	-6.0	0.0	1.0	‖	-80.0
3	‖ -0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-20.0
4	‖ -0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-8.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

pivotage autour de 1 et 6

Résultats de la transformation

Tableau "C" :

‖	2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	‖ 0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	-0.3	0.0	‖	20.0
2	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	-6.0	0.0	1.0	‖	-80.0
3	‖ -0.2	-0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-20.0
4	‖ -0.1	-0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	-8.0
5	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
6	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
7	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
8	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
9	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0
10	‖ 0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	‖	0.0

z = 0.00
w = 0.00

pivotage autour de 2 et 8

Résultats de la transformation

Tableau "C" :

‖	2000.0	3000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableau "D" :

‖	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

```
Tableaux "A" et "B"
 1 ||  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0 -0.3  0.0 ||  20.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0 -0.2 ||  13.3
 3 || -0.2 -0.4  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 || -20.0
 4 || -0.1 -0.1  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -8.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

  z =  0.00
  w =  0.00

pivotage autour de  3 et  2

   Résultats de la transformation

Tableau "C" :
  ||  500.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableau "D" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Tableaux "A" et "B"
 1 ||  0.0  0.0  0.0  0.0  0.0  1.0  0.0  0.0 -0.3  0.0 ||  20.0
 2 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.0  0.0 -0.2 ||  13.3
 3 ||  0.5  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  50.0
 4 || -0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||  -3.0
 5 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 6 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 7 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 8 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
 9 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0
10 ||  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 ||   0.0

  z = -150000.00
  w =  0.00

pivotage autour de  4 et  1

   Résultats de la transformation

Tableau "C" :
  ||   0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Tableau "D" :

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Tableaux "A" et "B"

1	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	-0.3	0.0	20.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-0.2	13.3
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	20.0
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	60.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

z = -180000.00
w = 0.00

Solution optimale :

Fonction objectif : z = -180000.00000

x(1)	=	60.00000
x(2)	=	20.00000
y(2,bruxelles)	=	20.00000
y(2,namur)	=	13.33333

Résultats sur les coûts :

x(1)	:	0.00000
x(2)	:	0.00000
y(1,bruxelles)	:	0.00000
y(1,liege)	:	0.00000
y(1,namur)	:	0.00000
y(2,bruxelles)	:	0.00000
y(2,liege)	:	0.00000
y(2,namur)	:	0.00000
un(liege),ve+	:	0.00000
un(namur),ve+	:	0.00000
deux,ve+	:	5000.00000
trois,ve+	:	10000.00000

Analyse de sensibilite sur la colonne 1 :

La solution reste valable pour le paramètre DELTA dans
[-500.00000,1000.00000]

avec une pente 60.00000

Analyse de sensibilite sur la colonne 2 :

La solution reste valable pour le paramètre DELTA dans
[-1000.00000,1000.00000]

avec une pente 20.00000

Analyse de sensibilite sur la colonne 3 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 4 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 5 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 6 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 20.00000

Analyse de sensibilite sur la colonne 7 :

La solution reste valable pour le paramètre DELTA dans
[0.00000,+∞]

avec une pente 0.00000

Analyse de sensibilite sur la colonne 8 :

La solution reste valable pour le paramètre DELTA dans
[-∞,+∞]

avec une pente 13.33333

Analyse de sensibilite sur la ligne 1 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 80.00000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 2 :

La solution reste valable pour le paramètre DELTA dans
[-∞, 80.00000]

avec une pente 0.00000

Analyse de sensibilite sur la ligne 3 :

La solution reste valable pour le paramètre DELTA dans
[-12.00000, 4.00000]

avec une pente -5000.00000

Analyse de sensibilite sur la ligne 4 :

La solution reste valable pour le paramètre DELTA dans
[-2.00000, 3.00000]

avec une pente -10000.00000

EXEMPLE DE L'INTRODUCTION

```
Fam : i = [1_10];
      j = [1_3];
      k = [1_5];
      t = [1_12];
      tp = [0_11];

Cst : #t, a(j, t) = 3, 4, 5;
      #t, #k, b(k, t) = 4;
      #t, b(3, t) = 3;
      #j, #t, c(j, k, t) = 3.2, 4.5, 6, 2.1, 4;
      #t, d(i, t) = 3, 2, 3, 2, 4, 2, 3, 3, 4, 4;
      d(4, 4) = 4;
      #i, h(i, t) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12;
      #i, h(i, 4) = 0;
      #i, init(i) = 0;
      init(3) = 1;
      init(5) = 3;
      init(7) = 2;
      #i, fin(i) = 1;
      #t, p(i, t) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10;
      #t, #i, s(i, t) = 0.85;


Equ : Max[ Som(i, t) p(i, t) * E(i, t) -
           Som(j, k, t) c(j, k, t) * W(j, k, t) -
           0.5 * Som(i) h(i, t><1) * varinit(i) -
           0.5 * Som(i, t<>1) h(i, t) * II(i, t><tp) -
           Som(i, t) 0.5 * h(i,t) * II(i, t) ];

      prod1, #j, #t, Som(k) W(j, k, t) - V(j,t) = 0;

      prod2, #j, #t, Som(i) U(i, t) - V(j, t) = 0;

      dispo1, #k, #t, Som(j) W(j, k, t) <= b(k, t);

      dispo2, #j, #t, V(j, t) <= a(j, t);

      equi1, #i, II(i,t><1) - U(i,t><1) + E(i,t><1) = init(i);

      equi2, #i, #t<>1, II(i,t)-II(i,t><tp)-U(i,t)+E(i,t) = 0;

      final, #i, II(i, t><12) = fin(i);

      initial, #i, varinit(i) = init(i);

      fraction, #i, #t, vars(i, t) = s(i, t);

      vente1, #i, #t, E(i, t) - d(i, t) * vars(i, t) >= 0;

      vente2, #i, #t, E(i, t) <= d(i, t);

Fin :


 5 familles :
   1 i
par intervalle :
de     1 a     10
```

```
  2 j
par intervalle :
de     1 a     3

  3 k
par intervalle :
de     1 a     5

  4 t
par intervalle :
de     1 a    12

  5 tp
par intervalle :
de     0 a    11


 9 constantes :
  1 a
  # :    1 t,  2 ,  3
fam :    1 j,  2 t,  3
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  5.0000  5.0000  5.0000
5.0000  5.0000  5.0000  5.0000  5.0000  5.0000  5.0000  5.0000  5.0000
  2 b
  # :    1 t,  2 k,  3
fam :    1 k,  2 t,  3
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
  3 c
  # :    1 j,  2 t,  3
fam :    1 j,  2 k,  3 t
3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000
3.2000  3.2000  3.2000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000
4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  6.0000  6.0000  6.0000
6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000
2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000
2.1000  2.1000  2.1000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  3.2000  3.2000  3.2000
3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000
4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000
4.5000  4.5000  4.5000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000
6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  2.1000  2.1000  2.1000
2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  3.2000  3.2000  3.2000  3.2000  3.2000  3.2000
3.2000  3.2000  3.2000  3.2000  3.2000  3.2000  4.5000  4.5000  4.5000
4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000  4.5000
6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000
6.0000  6.0000  6.0000  2.1000  2.1000  2.1000  2.1000  2.1000  2.1000
2.1000  2.1000  2.1000  2.1000  2.1000  2.1000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
```

```
  4 d
  # :    1 t,  2 ,  3
fam :    1 i,  2 t,  3
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000
2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
2.0000  2.0000  2.0000  4.0000  2.0000  2.0000  2.0000  2.0000  2.0000
2.0000  2.0000  2.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  2.0000  2.0000  2.0000
2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000
  5 h
  # :    1 i,  2 ,  3
fam :    1 i,  2 t,  3
1.0000  2.0000  3.0000  0.0000  5.0000  6.0000  7.0000  8.0000  9.0000
10.0000 11.0000 12.0000  1.0000  2.0000  3.0000  0.0000  5.0000  6.0000
7.0000  8.0000  9.0000 10.0000 11.0000 12.0000  1.0000  2.0000  3.0000
0.0000  5.0000  6.0000  7.0000  8.0000  9.0000 10.0000 11.0000 12.0000
1.0000  2.0000  3.0000  0.0000  5.0000  6.0000  7.0000  8.0000  9.0000
10.0000 11.0000 12.0000  1.0000  2.0000  3.0000  0.0000  5.0000  6.0000
7.0000  8.0000  9.0000 10.0000 11.0000 12.0000  1.0000  2.0000  3.0000
0.0000  5.0000  6.0000  7.0000  8.0000  9.0000 10.0000 11.0000 12.0000
1.0000  2.0000  3.0000  0.0000  5.0000  6.0000  7.0000  8.0000  9.0000
10.0000 11.0000 12.0000  1.0000  2.0000  3.0000  0.0000  5.0000  6.0000
7.0000  8.0000  9.0000 10.0000 11.0000 12.0000  1.0000  2.0000  3.0000
0.0000  5.0000  6.0000  7.0000  8.0000  9.0000 10.0000 11.0000 12.0000
1.0000  2.0000  3.0000  0.0000  5.0000  6.0000  7.0000  8.0000  9.0000
10.0000 11.0000 12.0000
  6 init
  # :    1 i,  2 ,  3
fam :    1 i,  2 ,  3
0.0000  0.0000  1.0000  0.0000  3.0000  0.0000  2.0000  0.0000  0.0000
0.0000
  7 fin
  # :    1 i,  2 ,  3
fam :    1 i,  2 ,  3
1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
1.0000
```

```
  8 p
  # :    1 t,  2 ,  3
fam :    1 i,  2 t,  3
1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000  1.0000
1.0000  1.0000  1.0000  2.0000  2.0000  2.0000  2.0000  2.0000  2.0000
2.0000  2.0000  2.0000  2.0000  2.0000  2.0000  3.0000  3.0000  3.0000
3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000  3.0000
4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000  4.0000
4.0000  4.0000  4.0000  5.0000  5.0000  5.0000  5.0000  5.0000  5.0000
5.0000  5.0000  5.0000  5.0000  5.0000  5.0000  6.0000  6.0000  6.0000
6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000  6.0000
7.0000  7.0000  7.0000  7.0000  7.0000  7.0000  7.0000  7.0000  7.0000
7.0000  7.0000  7.0000  8.0000  8.0000  8.0000  8.0000  8.0000  8.0000
8.0000  8.0000  8.0000  8.0000  8.0000  8.0000  9.0000  9.0000  9.0000
9.0000  9.0000  9.0000  9.0000  9.0000  9.0000  9.0000  9.0000  9.0000
10.0000 10.0000 10.0000 10.0000 10.0000 10.0000 10.0000 10.0000 10.0000
10.0000 10.0000 10.0000
  9 s
  # :    1 t,  2 i,  3
fam :    1 i,  2 t,  3
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500  0.8500
0.8500  0.8500  0.8500

Fonction objectif :     Il faut maximiser la fonction.
no_elem ! no_ent_som    !       code_som      !       val_som
  1 !     1     4     0 !     0     0     0 !     0     0     0
  2 !     2     3     4 !     0     0     0 !     0     0     0
  3 !     1     0     0 !     0     0     0 !     0     0     0
  4 !     1     4     0 !     0     1     0 !     0     1     0
  5 !     1     4     0 !     0     0     0 !     0     0     0

        !  cst          !       code_cst      !       val_cst
  1       8             !     0     0     0 !     0     0     0
  2       3             !     0     0     0 !     0     0     0
  3       5             !     0     1     0 !     0     1     0
  4       5             !     0     0     0 !     0     0     0
  5       5             !     0     0     0 !     0     0     0

        !  variable     !       code_var      !       val_var
  1       1             !     0     0     0 !     0     0     0
  2       2             !     0     0     0 !     0     0     0
  3       3             !     0     0     0 !     0     0     0
  4       4             !     0     3     0 !     0     5     0
  5       4             !     0     0     0 !     0     0     0
```

```
mult :
  1     1.000
  2    -1.000
  3    -0.500
  4    -0.500
  5    -0.500


Equations :
   equation    1  nom : "prod1"
      no_ent_ptt          code_ptt          val_ptt
    2     4     0 !     0     0     0 !     0     0     0
Code =    1

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      3     0     0 !      0     0     0 !      0     0     0
  2  !      0     0     0 !      0     0     0 !      0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !      0     0     0 !      0     0     0
  2  !      0             !      0     0     0 !      0     0     0

             var1          !     code1_var       !     val1_var
  1  !      2             !      0     0     0 !      0     0     0
  2  !      5             !      0     0     0 !      0     0     0

    mult1
  1  1.00000
  2 -1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2          !     code2_cst       !     val2_cst
  1  !      0             !      0     0     0 !      0     0     0

    mult2
  1  0.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
   equation    2  nom : "prod2"
      no_ent_ptt          code_ptt          val_ptt
    2     4     0 !     0     0     0 !     0     0     0
Code =    1
```

```
1er membre :

                no1_ent_som    !     code1_som       !     val1_som
  1   !      1      0      0 !      0      0      0 !      0      0      0
  2   !      0      0      0 !      0      0      0 !      0      0      0

                 cst1          !     code1_cst       !     val1_cst
  1   !      0                 !      0      0      0 !      0      0      0
  2   !      0                 !      0      0      0 !      0      0      0

                 var1          !     code1_var       !     val1_var
  1   !      6                 !      0      0      0 !      0      0      0
  2   !      5                 !      0      0      0 !      0      0      0

    mult1
  1  1.00000
  2 -1.00000

2nd membre :

                no2_ent_som    !     code2_som       !     val2_som
  1   !      0      0      0 !      0      0      0 !      0      0      0

                 cst2          !     code2_cst       !     val2_cst
  1   !      0                 !      0      0      0 !      0      0      0

    mult2
  1  0.00000

Variables :
"e"    1,   4,   0,
"w"    2,   3,   4,
"varinit"    1,   0,   0,
"ii"    1,   4,   0,
"v"    2,   4,   0,
"u"    1,   4,   0,
"vars"    1,   4,   0,
   equation    3  nom : "dispo1"
      no_ent_ptt          code_ptt          val_ptt
    3      4      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

                no1_ent_som    !     code1_som       !     val1_som
  1   !      2      0      0 !      0      0      0 !      0      0      0

                 cst1          !     code1_cst       !     val1_cst
  1   !      0                 !      0      0      0 !      0      0      0

                 var1          !     code1_var       !     val1_var
  1   !      2                 !      0      0      0 !      0      0      0

    mult1
  1  1.00000
```

```
2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst2          !     code2_cst       !     val2_cst
  1  !      2             !      0      0      0 !      0      0      0

     mult2
  1  1.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
   equation    4  nom : "dispo2"
      no_ent_ptt          code_ptt            val_ptt
    2      4      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !      0      0      0 !      0      0      0

             var1          !     code1_var       !     val1_var
  1  !      5             !      0      0      0 !      0      0      0

     mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst2          !     code2_cst       !     val2_cst
  1  !      1             !      0      0      0 !      0      0      0

     mult2
  1  1.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
```

```
   equation   5  nom : "equi1"
      no_ent_ptt          code_ptt          val_ptt
    1     0     0 !     0     0     0 !     0     0     0
Code =    1

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1   !     0     0     0 !     0     0     0 !     0     0     0
  2   !     0     0     0 !     0     0     0 !     0     0     0
  3   !     0     0     0 !     0     0     0 !     0     0     0

             cst1          !     code1_cst       !     val1_cst
  1   !     0             !     0     0     0 !     0     0     0
  2   !     0             !     0     0     0 !     0     0     0
  3   !     0             !     0     0     0 !     0     0     0

             var1          !     code1_var       !     val1_var
  1   !     4             !     0     1     0 !     0     1     0
  2   !     6             !     0     1     0 !     0     1     0
  3   !     1             !     0     1     0 !     0     1     0

    mult1
  1  1.00000
  2 -1.00000
  3  1.00000

 2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1   !     0     0     0 !     0     0     0 !     0     0     0

             cst2          !     code2_cst       !     val2_cst
  1   !     6             !     0     0     0 !     0     0     0

    mult2
  1  1.00000

 Variables :
 "e"    1,   4,   0,
 "w"    2,   3,   4,
 "varinit"    1,   0,   0,
 "ii"    1,   4,   0,
 "v"    2,   4,   0,
 "u"    1,   4,   0,
 "vars"    1,   4,   0,
    equation   6  nom : "equi2"
       no_ent_ptt          code_ptt          val_ptt
     1     4     0 !     0     1     0 !     0     1     0
 Code =    1

 1er membre :

            no1_ent_som    !     code1_som       !     val1_som
   1   !     0     0     0 !     0     0     0 !     0     0     0
   2   !     0     0     0 !     0     0     0 !     0     0     0
   3   !     0     0     0 !     0     0     0 !     0     0     0
   4   !     0     0     0 !     0     0     0 !     0     0     0
```

```
          cst1          !     code1_cst       !     val1_cst
1  !      0             !     0     0     0   !     0     0     0
2  !      0             !     0     0     0   !     0     0     0
3  !      0             !     0     0     0   !     0     0     0
4  !      0             !     0     0     0   !     0     0     0

          var1          !     code1_var       !     val1_var
1  !      4             !     0     0     0   !     0     0     0
2  !      4             !     0     3     0   !     0     5     0
3  !      6             !     0     0     0   !     0     0     0
4  !      1             !     0     0     0   !     0     0     0

   mult1
1   1.00000
2  -1.00000
3  -1.00000
4   1.00000

2nd membre :

          no2_ent_som   !     code2_som       !     val2_som
1  !      0     0     0  !     0     0     0   !     0     0     0

          cst2          !     code2_cst       !     val2_cst
1  !      0             !     0     0     0   !     0     0     0

   mult2
1   0.00000

Variables :
"e"    1,   4,   0,
"w"    2,   3,   4,
"varinit"    1,   0,   0,
"ii"    1,   4,   0,
"v"    2,   4,   0,
"u"    1,   4,   0,
"vars"    1,   4,   0,
   equation    7   nom : "final"
      no_ent_ptt          code_ptt          val_ptt
    1     0     0  !     0     0     0   !     0     0     0
Code =    1

1er membre :

          no1_ent_som   !     code1_som       !     val1_som
1  !      0     0     0  !     0     0     0   !     0     0     0

          cst1          !     code1_cst       !     val1_cst
1  !      0             !     0     0     0   !     0     0     0

          var1          !     code1_var       !     val1_var
1  !      4             !     0     1     0   !     0    12     0

   mult1
1   1.00000
```

```
2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1   !     0      0      0 !     0      0      0 !     0      0      0

             cst2          !     code2_cst       !     val2_cst
  1   !     7             !     0      0      0 !     0      0      0

    mult2
  1  1.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
   equation    8  nom : "initial"
      no_ent_ptt          code_ptt          val_ptt
    1     0      0 !     0      0      0 !     0      0      0
Code =    1

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1   !     0      0      0 !     0      0      0 !     0      0      0

             cst1          !     code1_cst       !     val1_cst
  1   !     0             !     0      0      0 !     0      0      0

             var1          !     code1_var       !     val1_var
  1   !     3             !     0      0      0 !     0      0      0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1   !     0      0      0 !     0      0      0 !     0      0      0

             cst2          !     code2_cst       !     val2_cst
  1   !     6             !     0      0      0 !     0      0      0

    mult2
  1  1.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
```

```
   equation   9  nom : "fraction"
      no_ent_ptt          code_ptt          val_ptt
    1     4     0 !     0     0     0 !     0     0     0
Code =   1

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !      0     0     0 !      0     0     0

             var1          !     code1_var       !     val1_var
  1  !      7             !      0     0     0 !      0     0     0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0     0     0 !      0     0     0 !      0     0     0

             cst2          !     code2_cst       !     val2_cst
  1  !      9             !      0     0     0 !      0     0     0

    mult2
  1  1.00000

Variables :
"e"   1,  4,  0,
"w"   2,  3,  4,
"varinit"   1,  0,  0,
"ii"   1,  4,  0,
"v"   2,  4,  0,
"u"   1,  4,  0,
"vars"   1,  4,  0,
   equation  10  nom : "vente1"
      no_ent_ptt          code_ptt          val_ptt
    1     4     0 !     0     0     0 !     0     0     0
Code =   2

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      0     0     0 !      0     0     0 !      0     0     0
  2  !      0     0     0 !      0     0     0 !      0     0     0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !      0     0     0 !      0     0     0
  2  !      4             !      0     0     0 !      0     0     0

             var1          !     code1_var       !     val1_var
  1  !      1             !      0     0     0 !      0     0     0
  2  !      7             !      0     0     0 !      0     0     0

    mult1
  1  1.00000
  2 -1.00000
```

```
2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst2          !     code2_cst       !     val2_cst
  1  !      0             !      0      0      0 !      0      0      0

    mult2
  1  0.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,
   equation  11  nom : "vente2"
      no_ent_ptt         code_ptt          val_ptt
    1      4      0 !      0      0      0 !      0      0      0
Code =    3

1er membre :

            no1_ent_som    !     code1_som       !     val1_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst1          !     code1_cst       !     val1_cst
  1  !      0             !      0      0      0 !      0      0      0

             var1          !     code1_var       !     val1_var
  1  !      1             !      0      0      0 !      0      0      0

    mult1
  1  1.00000

2nd membre :

            no2_ent_som    !     code2_som       !     val2_som
  1  !      0      0      0 !      0      0      0 !      0      0      0

             cst2          !     code2_cst       !     val2_cst
  1  !      4             !      0      0      0 !      0      0      0

    mult2
  1  1.00000

Variables :
"e"    1,  4,  0,
"w"    2,  3,  4,
"varinit"    1,  0,  0,
"ii"    1,  4,  0,
"v"    2,  4,  0,
"u"    1,  4,  0,
"vars"    1,  4,  0,

*** erreur (72) du type execution tableau du simplexe trop grand ***
```